# *Declarative Machine Learning and Data Mining*

Luc De Raedt

*CP for Analytics Workshop, Cork, 2015*

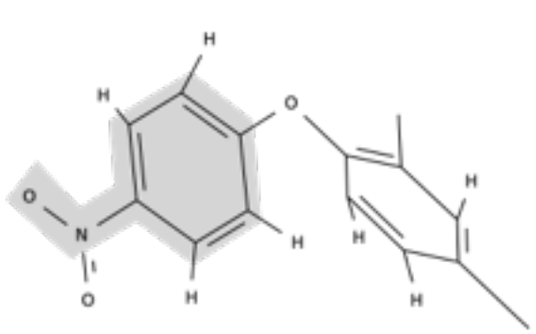
*Joint work with Angelika Kimmig, Siegfried Nijssen, Tias Guns and many others*

# Motivation

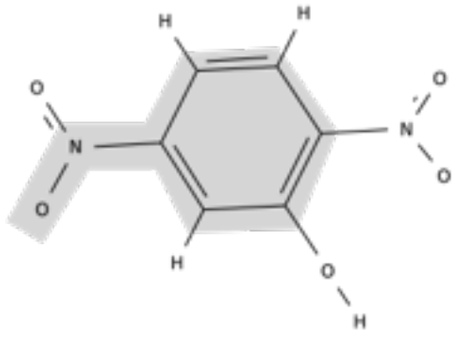
- Use of declarative methods, a new trend in machine learning and data mining
- Two sources of inspiration
  - Stephen Boyd on Convex Optimisation for Statistical Learning
  - Helmut Simonis in CP
- Introduce two types of declarative languages for ML & DM, logic / symbolic oriented rather than pure linear algebra
  - CP4IM & MiningZinc — CP for pattern mining
  - Probabilistic Programming — in ProbLog

# Declarative Mining

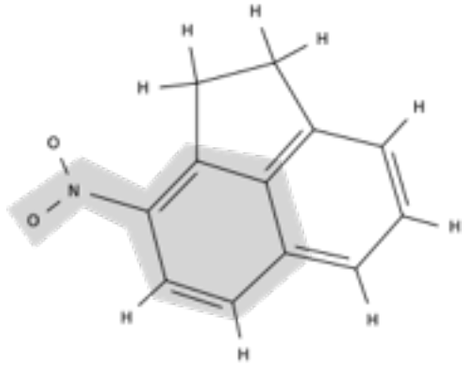
# Patterns



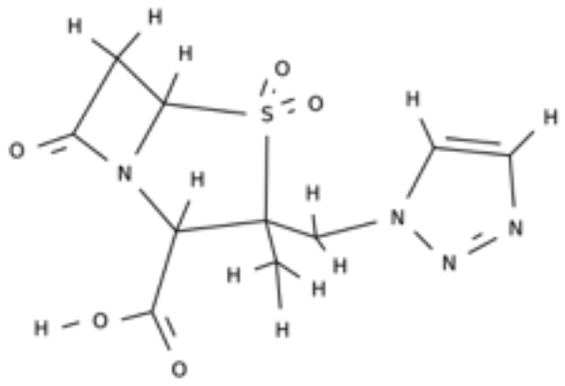
Mutagenic



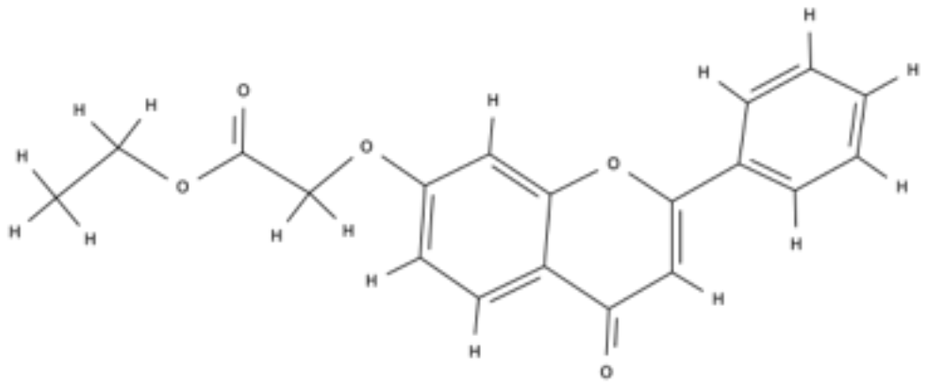
Mutagenic



Mutagenic



Clean



Clean

# Pattern Mining

## A. frequent pattern

- which patterns are frequent ?

$$Th(\mathcal{L}, Q, \mathcal{D}) = \{p \in \mathcal{L} | Q(p, \mathcal{D}) = true\}$$

## B. Correlated pattern mining = subgroup discovery

- which patterns are significant w.r.t. classes ? all patterns ? k-best patterns ?

$$Th(\mathcal{L}, Q, \mathcal{D}) = \arg_{p \in \mathcal{L}} \max_k \phi(p, \mathcal{D})$$

## C. pattern set mining

- which pattern *set* is the *best* concept-description for the actives ? for the inactives ?

$$Th(\mathcal{L}, \mathcal{Q}, \mathcal{D}) = \{P \subseteq \mathcal{L} | \mathcal{Q}(P, \mathcal{D}) = true\}$$

# Itemset mining

Data

Frequent patterns

4

2

3

# A. Frequent Itemset Mining

Given

- $\mathcal{I} = \{1, \cdots, NrI\}$
  set of items
- $\mathcal{T} = \{1, \cdots, NrT\}$
  set of transactions identifiers
- $\mathcal{D} = \{(t, I) | t \in \mathcal{T}, I \subseteq \mathcal{I}\}$
  Dataset
- $Items \subseteq \mathcal{I}$ and $Trans \subseteq \mathcal{T}$

Find $Items$ such that
$|covers(Items, \mathcal{D})| > freq$

where $covers(Items, \mathcal{D}) =$
$\{t \in \mathcal{T} | (t, I) \in \mathcal{D}$ and $Items \subseteq I\}$

# A.Frequent Itemset Mining

Given

- $\mathcal{I} = \{1, \cdots, NrI\}$
  set of items
- $\mathcal{T} = \{1, \cdots, NrT\}$
  set of transactions identifiers
- $\mathcal{D} = \{(t, I) | t \in \mathcal{T}, I \subseteq \mathcal{I}\}$
  Dataset
- $Items \subseteq \mathcal{I}$ and $Trans \subseteq \mathcal{T}$

Find $Items$ such that
$|covers(Items, \mathcal{D})| > freq$

where $covers(Items, \mathcal{D}) =$
$\{t \in \mathcal{T} | (t, I) \in \mathcal{D}$ and $Items \subseteq I\}$

```
int: Freq;
int: NrI;
int: NrT;

array[1..NrT] of set of 1..NrI: D;

var set of 1..NrI: Items;
var set of 1..NrT: Trans;

constraint card(Trans) > Freq;
constraint Trans = covers(Items, D);

solve satisfy;


function var set of int: cover(Items, D) =
let {
        var set of int: Trans,
        constraint forall (t in ub(Trans))
        (t in Trans ↔ Items subset D[t])
} in Trans;
```

# Frequent Itemset Mining

math like notation

user defined functions and constraints

solver independent (standardized)

efficiently solvable

```
int: Freq;
int: NrI;
int: NrT;

array[1..NrT] of set of 1..NrI: D;

var set of 1..NrI: Items;
var set of 1..NrT: Trans;

constraint card(Trans) > Freq;
constraint Trans = covers(Items, D);

solve satisfy;

function var set of int: cover(Items, D) =
let {
        var set of int: Trans,
        constraint forall (t in ub(Trans))
        (t in Trans ↔ Items subset D[t])
} in Trans;
```

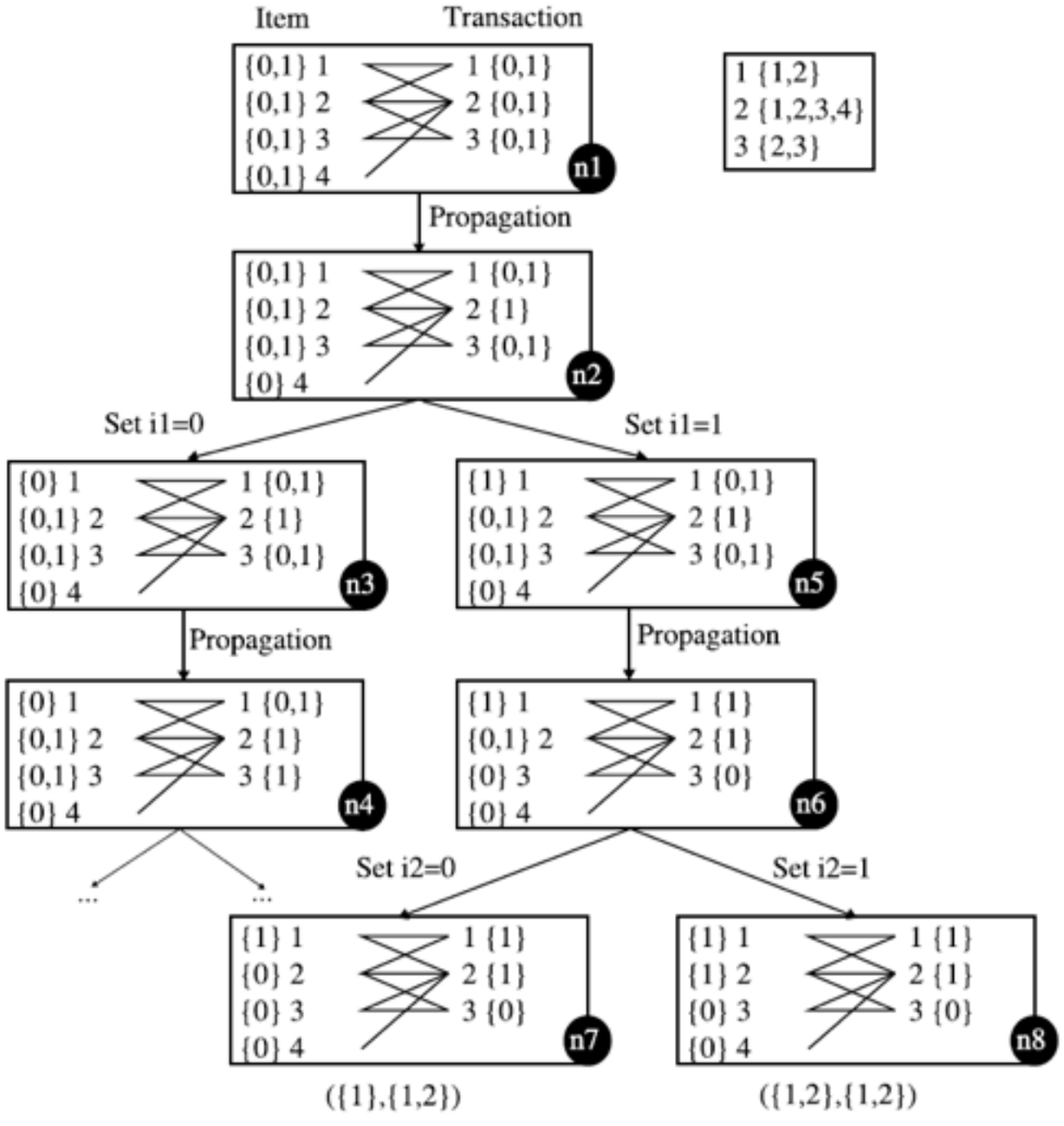
# Encoding in Zinc

```
int: Freq;
int: NrI; int: NrT;
array [1..NrT] of set of int: D;

array [1..NrI] of var bool: Items;
array [1..NrT] of var bool: Trans;

constraint % encode D: every Trans complement has no supported Items
  forall(t in 1..NrT) (
     Trans[t] <-> sum(i in 1..NrI) ( Items[i]*(1 - (i in D[t])) ) <= 0
  );

constraint % frequency: every Item is supported by sufficently many Trans
  forall(i in 1..NrI) (
     Items[i] -> sum(t in 1..NrT) ( Trans[t]*(i in D[t]) ) >= Freq
  );

solve satisfy;
```

$$\forall t : T_t = 1 \Leftrightarrow \sum_i I_i(1 - D_{ti}) = 0$$

$$\sum_t T_t \geq minsup \quad \textbf{iff} \quad \forall i : I_i = 1 \Rightarrow \sum_t T_t D_{ti} \geq minsup$$

# Resulting Search Strategy akin to Zaki's Eclat [KDD 97]



# Closed Itemset Mining

```
int: Freq;
int: NrI;
int: NrT;

array[1..NrT] of set of 1..NrI: D;

var set of 1..NrI: Items;
var set of 1..NrT: Trans;
constraint card(Trans) > Freq;
constraint Trans = covers(Items, D);
constraint Items = cover_inv(Trans, D);
solve satisfy;
```

```
function var set of int: cover_inv(Trans,D)=
 let {
        var set of int: Items,
        constraint forall (i in ub(Items))
      (i in Items ↔ Trans subset D'[i] )
} in Items;
```

```
function var set of int: cover(Items, D) =
let {
        var set of int: Trans,
        constraint forall (t in ub(Trans))
      (t in Trans ↔ Items subset D[t] )
} in Trans;
```

# Further Constraints

* exact coverage :
  t in Trans <-> Items subset D[t]

* freq:
  i in Items -> card(Trans intersect D'[i]) >= Freq

* maximal:
  i in Items <-> card(Trans intersect D'[i]) >= Freq

* closed:
  i in Items <-> Trans subset D'[i]

* delta-closed:
  i in Items <-> card(Trans intersect D'[i]) <= Delta*card(Trans)

# Top-k Correlated Pattern Mining

- $\mathcal{D}$ now consists of two datasets, say $P$ and $N$
- a correlation function $\phi(p, \mathcal{D})$, e.g., $\chi^2$
- $Th(\mathcal{L}, Q, \mathcal{D}) = \arg_{p \in \mathcal{L}} \max_k \phi(p, \mathcal{D})$

# Modeling perspective

```
int: NrI; int: NrT; int: Freq;
array[1..NrT] of set of int: D;
set of int: pos; set of int: neg;

var set of 1..NrI: Items;
var set of 1..NrT: Trans;

constraint Trans = cover(Items, D);
constraint Items = cover_inv(Trans, D);

solve maximize
        card(Trans intersect pos) – card(Trans intersect neg)
```

accuracy

Alternative opt. functions, for example:

```
solve maximize chi2(Trans, pos, neg);
```

with:

```
function float: chi2(Trans, pos, neg)
```

# Correlation function

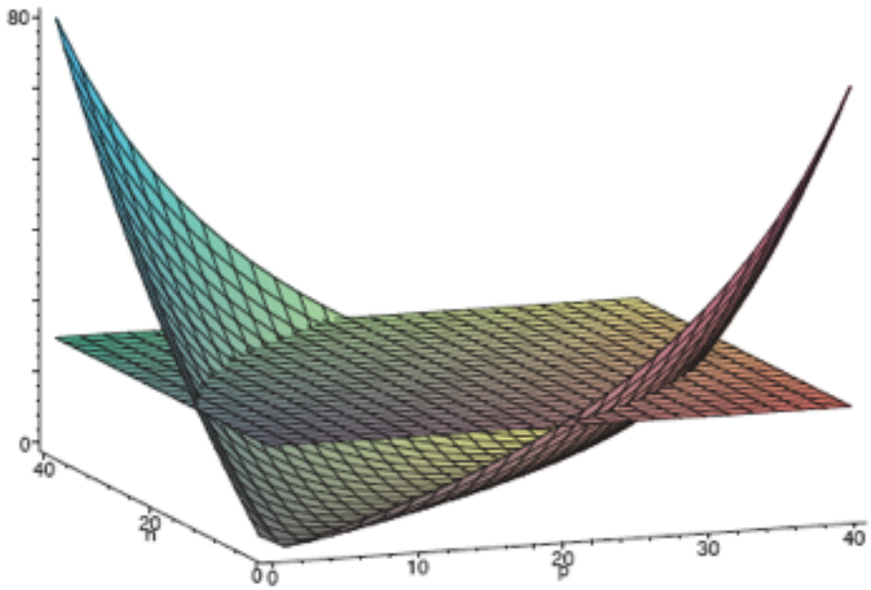
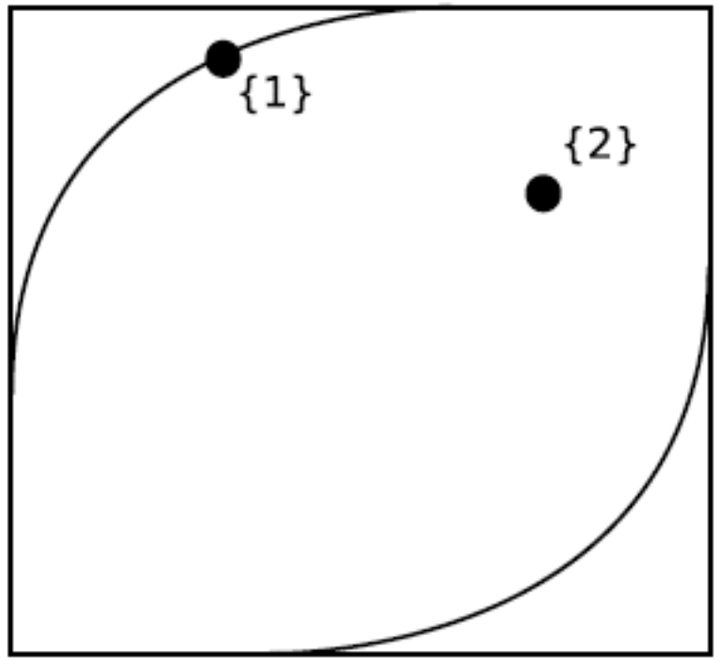



**Figure 1: A plot of the $\chi^2$ scoring function, and a threshold on $\chi^2$.**



# Generality

| | LCM [15] | MAFIA [6] | ExAMiner [4] | DualMiner [5] | DDPrune [12] | CP4IM |
|---|---|---|---|---|---|---|
| Constraints on data | | | | | | |
| Minimum frequency | X | X | X | X | | X |
| Maximum frequency | | | | X | | X |
| Emerging patterns | | | | | | X |
| Condensed Representations | | | | | | |
| Maximal | X | X | | X | | X |
| Closed | X | X | | | | X |
| $\delta-$Closed | | | | | | X |
| Constraints on syntax | | | | | | |
| Max/Min total cost | | | X | X | | X |
| Minimum average cost | | | X | | | X |
| Max/Min size | X | X | X | X | | X |
| Constraints on labelled data | | | | | | |
| Minimum correlation | | | | | X | X |
| Maximum correlation | | | | | | X |

[L. De Raedt, T. Guns, S. Nijssen, AAAI 2010]

# Declarative constraint-based mining

Language goals:

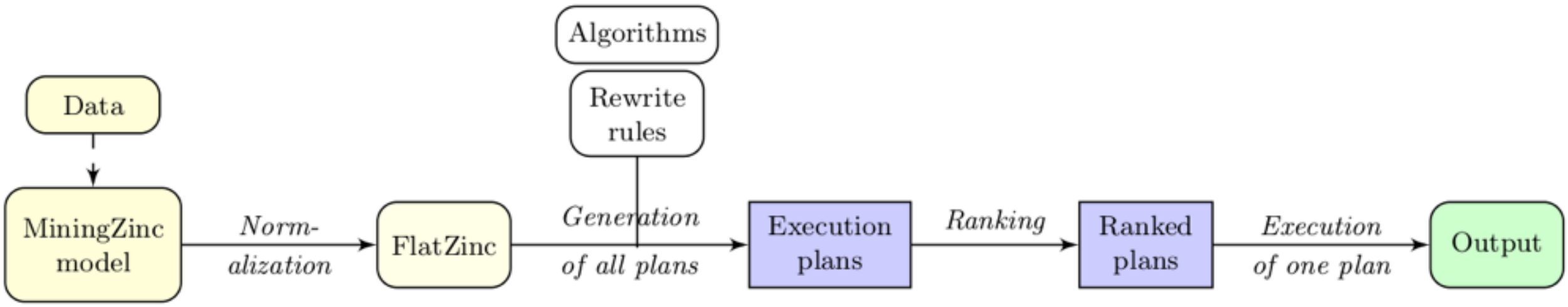
- high-level notation (similar to paper definitions)
- solver-independent
- user-defined abstractions


- mathematical-like language (Zinc)
- many solvers
- can define custom predicates *and functions*

=> MiningZinc

[T. Guns, A. Dries, S. Nijssen, G. Tack, L. De Raedt, IJCAI 2013]

# Toolchain




1) Normalize to FlatZinc *(do not flatten lib_itemsetmining.mzn yet)*
2) Apply rewrite rules to:
   - add redundant constraints
   - detect (partial) applicability of specialised algorithms
   - tailor to constraint solvers
3) Collect all feasible rewrite combinations = execution plans
4) Heuristically rank + execute a plan

```
var set of 1..NrI: Items; array[int] of set of int: TDB;
constraint card(cover(Items, TDB)) >= 20;
constraint card(cover(Items, TDB)) =< 40;
solve satisfy;
```

Three categories of execution plans:

A) Specialised algorithms only

- Eclat-maxfreq(TDB, 20, 50)
- LCMv2(TDB, 20) + maxcover(Items, TDB, 40)

B) Hybrid decomposition

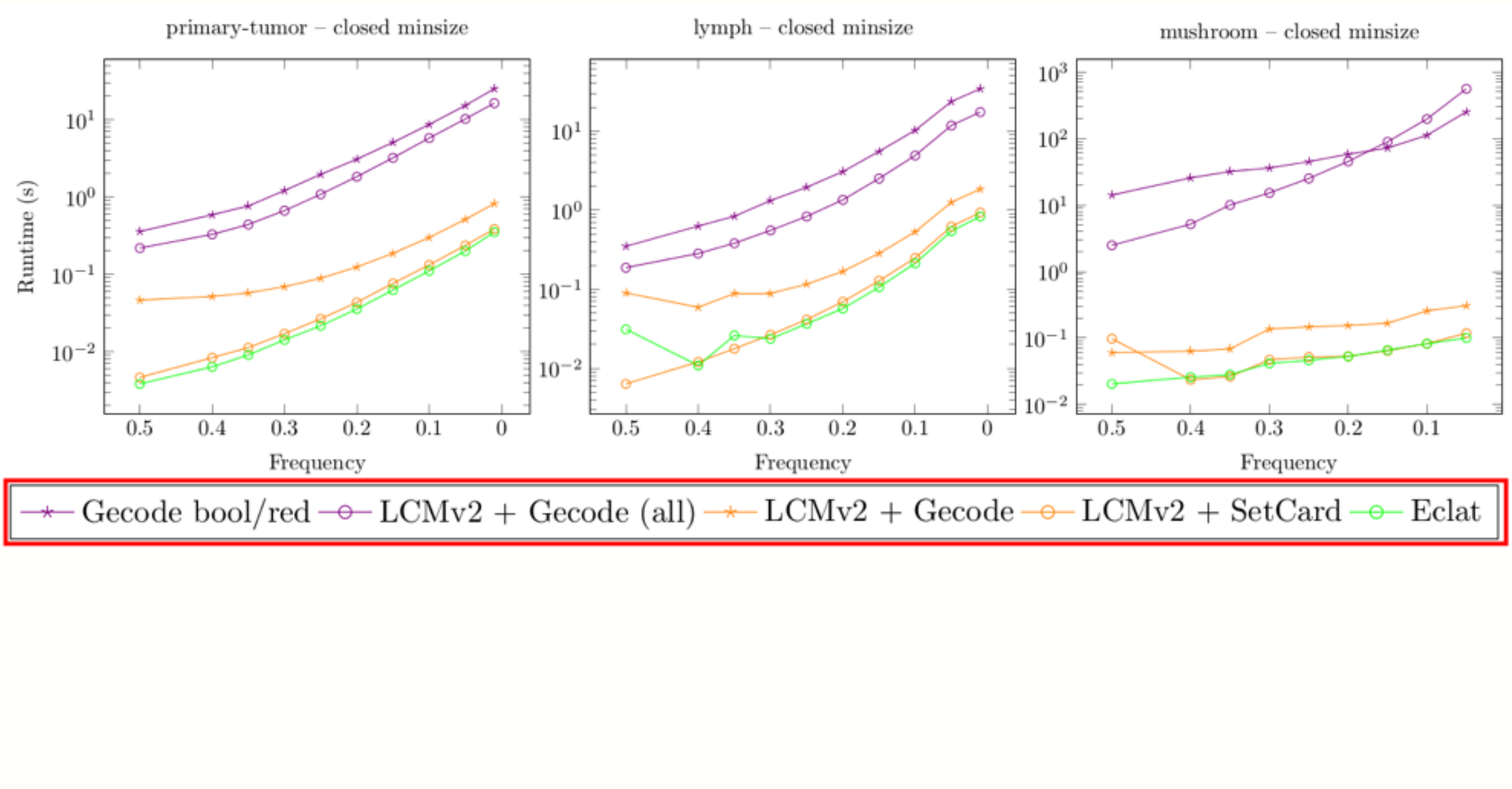
- LCMv2(TDB, 20) + gecode(card(cover(Items, TDB)) =< 40)
- LCMv2(TDB, 20) + frequency(Items, TDB, S) + gecode(S =< 40)

C) Generic solvers only

- gecode(...)
- gecode-bool(...)
- gecode-bool(... *+ redundant*)
- or-tools-bool(...)

# Experiments, hybrid solving

frequent itemset mining, with minimum size and closure constraint



https://dtai.cs.kuleuven.be/CP4IM/download.php

Apps | Myroute | My Touring Mobilis | Street Photography | Save to Mendeley | Google Scholar | dupong

CP4IM

# Constraint Programming for Itemset Mining

Home | Software | Datasets | Publications

## Software

**MiningZinc**, this high-level declarate framework for constraint-based mining is available on a separate page.

### Download FIM_CP

The frequent and constraint-based itemset mining system using Constraint Programming.

- Publication: *L. De Raedt, T. Guns, S. Nijssen. Constraint programming for itemset mining, KDD 2008.*
- Publication: *T. Guns, S. Nijssen, L. De Raedt. Itemset mining: A constraint programming perspective, Artificial Intelligence 175(12-13), 2011.*
- **Latest version:** fimcp-2.7.tar.gz
- License: MIT license
- Language: C++ (tested on Linux, known to work under Mac and Windows)
- Latest changes: Release with new, easier, build system: it now automatically downloads, configures and compiles the Gecode CP solver (version 3.7.1).

Installation instructions in the accompagnying README. See online usage instructions and examples.

inductiveconstraints.eu/miningzinc/#

Click to go back, hold to see history

uring Mobilis | Street Photography | Save to Mendeley | Google Scholar | dupong

MiningZinc | Basic usage | Modeling | Examples | Download

# Basic usage

MiningZinc can be used through two interfaces: as a command line tool and as a Python package.

## From the command line

MiningZinc can be run through its command line interface. There are four modes:

- `list` : analyze the model and list the possible execution plans
- `solve` : solve the model
- `interactive` : combination of previous two modes with a choice menu
- default: solve the model using the first available strategy

The default usage of MiningZinc is:

```
./miningzinc model.mzn data1.dzn data2.dzn -D "Param1=10;"
```

# State of the art

- Many other works
  - Clustering (Ian Davidson, Christel Vrain, Thi-Bich-Hanh Dao, …)
  - Skyline patterns (B. Cremilleux, … ; Negrevergne, … )
  - Other pattern types (L. Sais, ..; Guns, … )
- Challenges
  - Scaling
  - Structured data (graphs, sequences, …)
  - Integration with some other ML/DM techniques
  - Other types of solvers …

# Probabilistic Programming

# World Dynamics

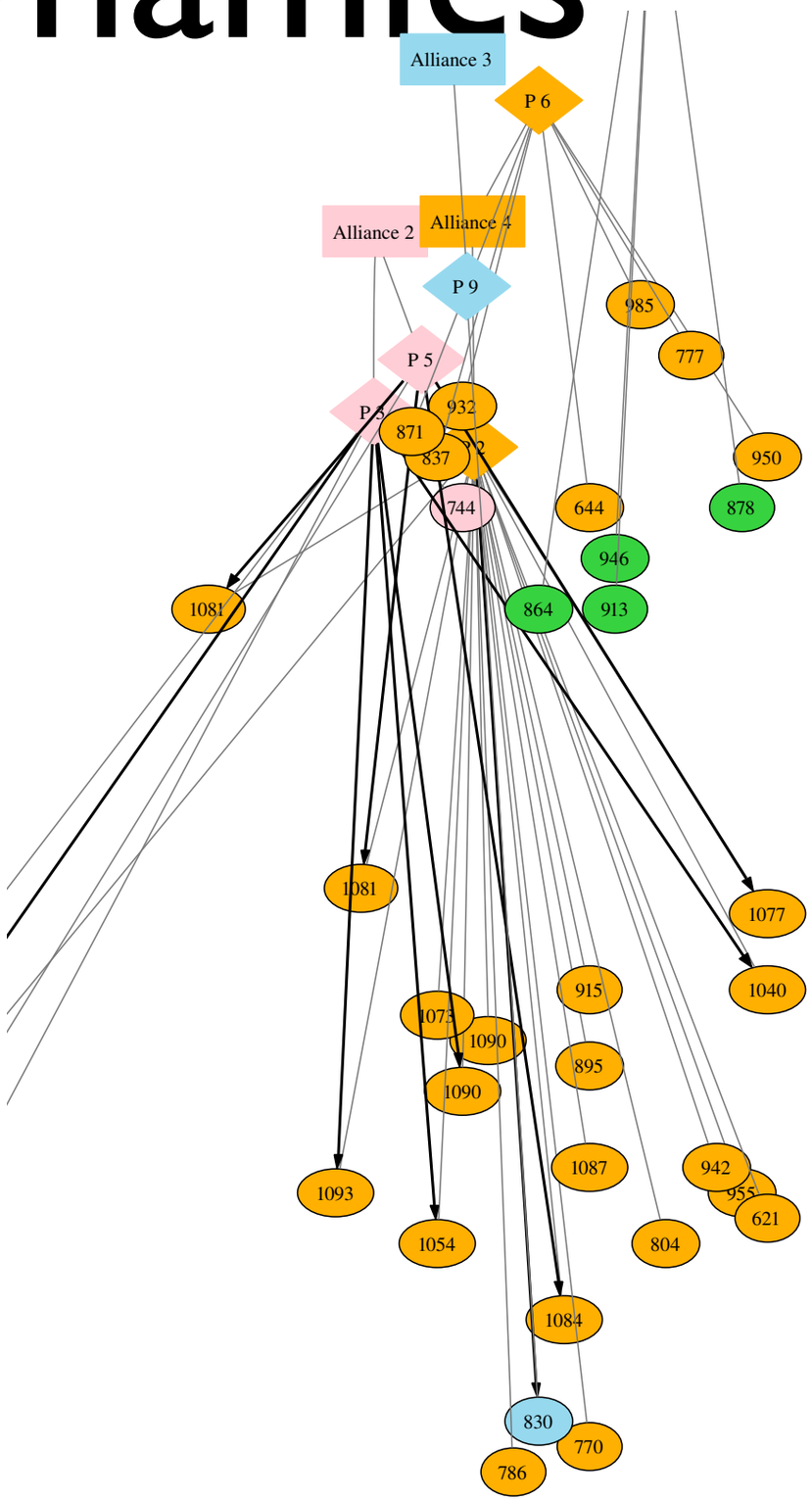
Fragment of world with

~10 alliances
~200 players
~600 cities

alliances color-coded

Can we build a model
of this world ?
Can we use it for playing
better ?

[Thon, Landwehr, De Raedt, ECML08]



# World Dynamics

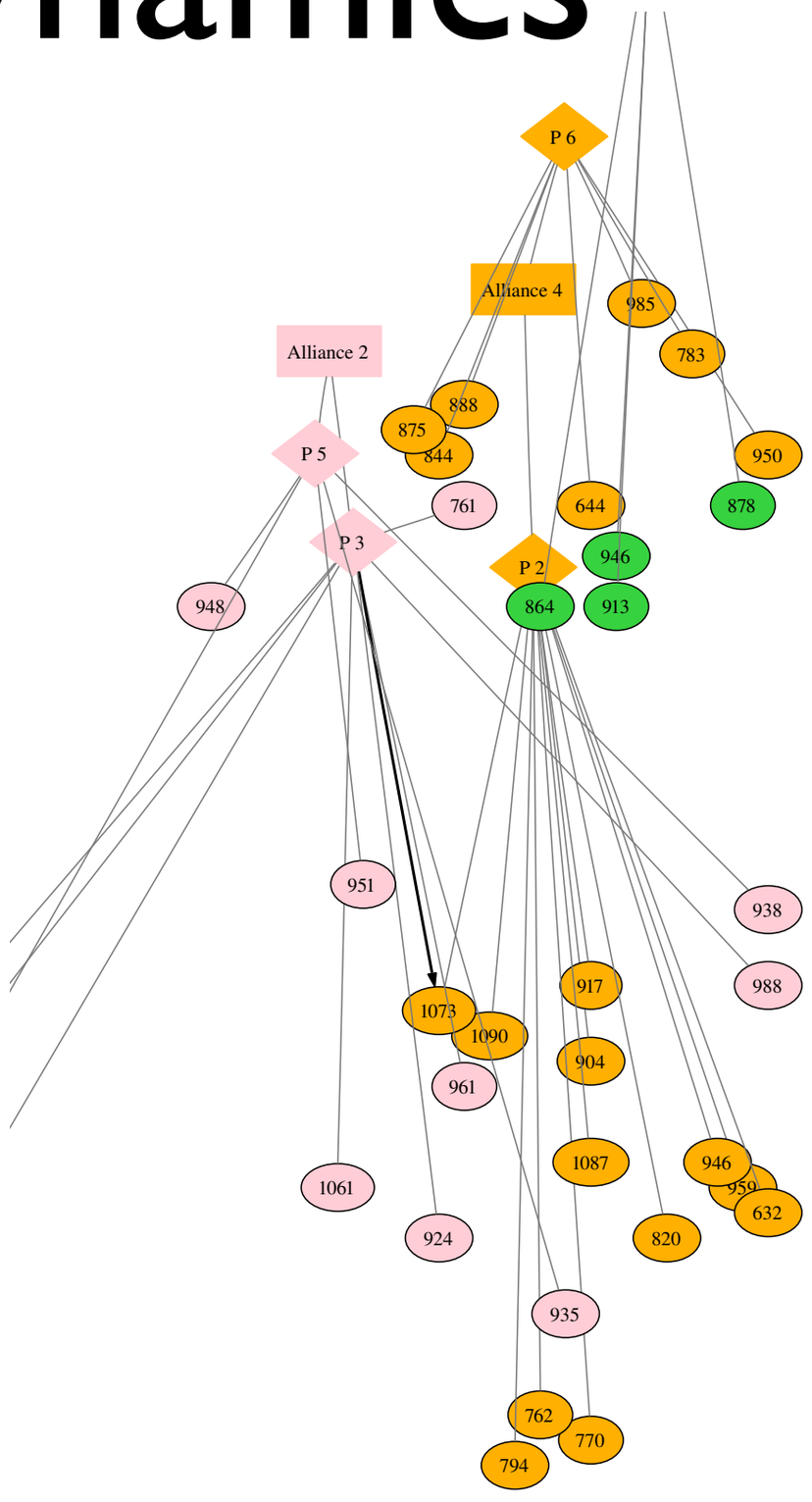
Fragment of world with

~10 alliances
~200 players
~600 cities

alliances color-coded

Can we build a model
of this world ?
Can we use it for playing
better ?

[Thon, Landwehr, De Raedt, ECML08]



# World Dynamics

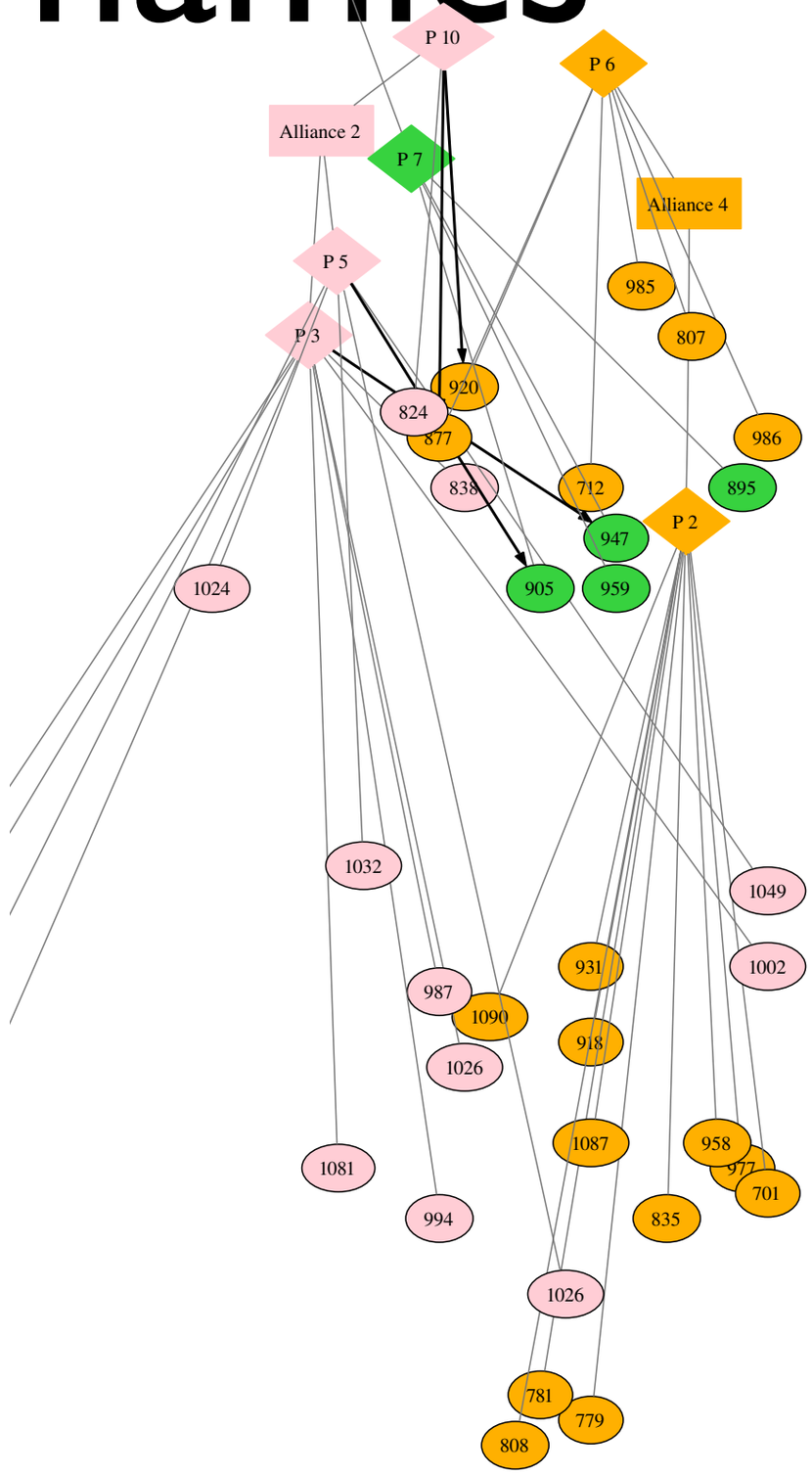
Fragment of world with

~10 alliances
~200 players
~600 cities

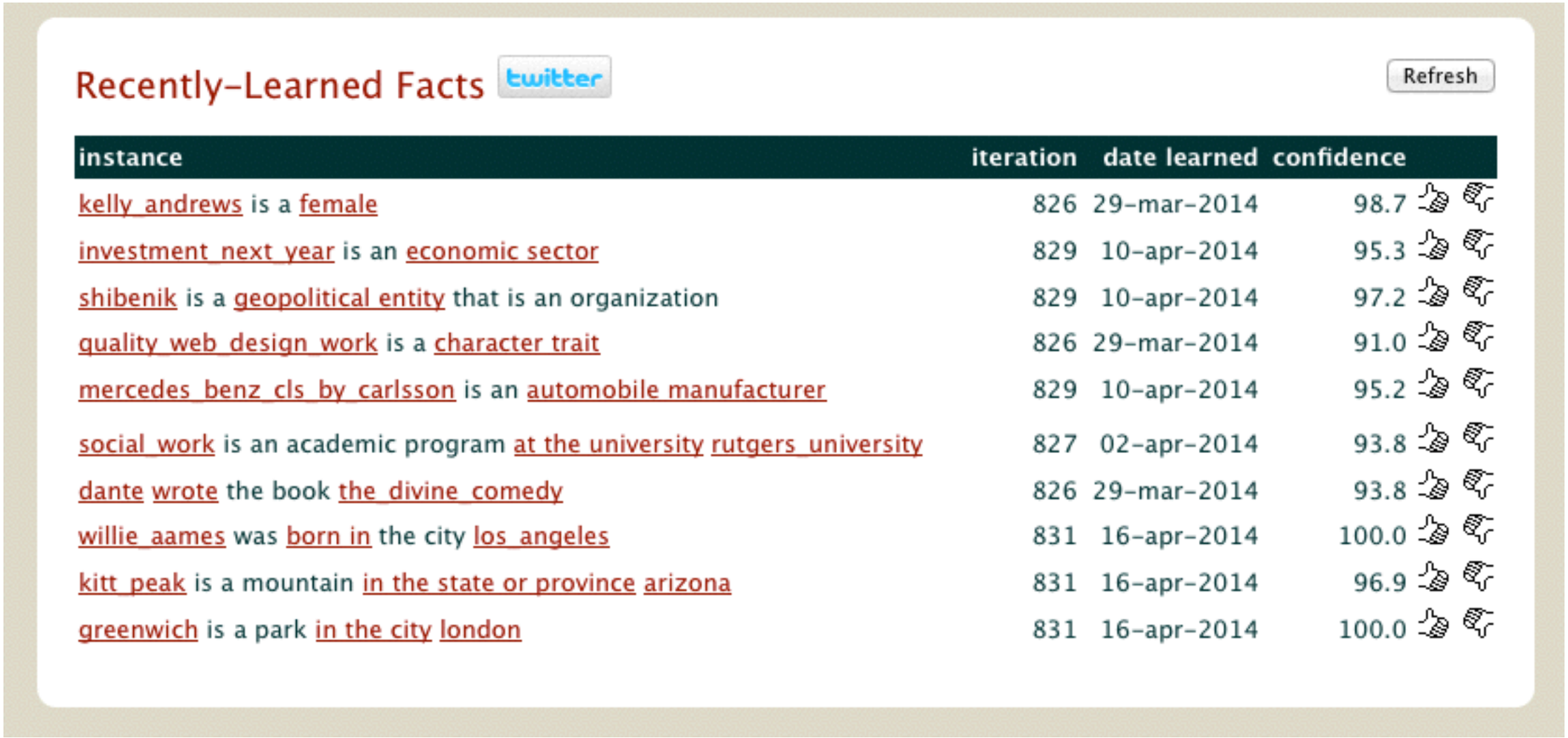
alliances color-coded

Can we build a model
of this world ?
Can we use it for playing
better ?

[Thon, Landwehr, De Raedt, ECML08]



# Example: Information Extraction



Recently-Learned Facts twitter

Refresh

| instance | iteration | date learned | confidence |
|---|---|---|---|
| kelly_andrews is a female | 826 | 29-mar-2014 | 98.7 |
| investment_next_year is an economic sector | 829 | 10-apr-2014 | 95.3 |
| shibenik is a geopolitical entity that is an organization | 829 | 10-apr-2014 | 97.2 |
| quality_web_design_work is a character trait | 826 | 29-mar-2014 | 91.0 |
| mercedes_benz_cls_by_carlsson is an automobile manufacturer | 829 | 10-apr-2014 | 95.2 |
| social_work is an academic program at the university rutgers_university | 827 | 02-apr-2014 | 93.8 |
| dante wrote the book the_divine_comedy | 826 | 29-mar-2014 | 93.8 |
| willie_aames was born in the city los_angeles | 831 | 16-apr-2014 | 100.0 |
| kitt_peak is a mountain in the state or province arizona | 831 | 16-apr-2014 | 96.9 |
| greenwich is a park in the city london | 831 | 16-apr-2014 | 100.0 |



NELL: http://rtw.ml.cmu.edu/rtw/

# Example: Information Extraction

Recently-Learned Facts twitter Refresh

| instance | iteration | date learned | confidence |
|---|---|---|---|
| kelly_andrews is a female | 826 | 29-mar-2014 | 98.7 |
| investment_next_year is an economic sector | 829 | 10-apr-2014 | 95.3 |
| shibenik is a geopolitical entity that is an organization | 829 | 10-apr-2014 | 97.2 |
| quality_web_design_work is a character trait | 826 | 29-mar-2014 | 91.0 |
| mercedes_benz_cls_by_carlsson is an automobile manufacturer | 829 | 10-apr-2014 | 95.2 |
| social_work is an academic program at the university rutgers_university | 827 | 02-apr-2014 | 93.8 |
| dante wrote the book the_divine_comedy | 826 | 29-mar-2014 | 93.8 |
| willie_aames was born in the city los_angeles | 831 | 16-apr-2014 | 100.0 |
| kitt_peak is a mountain in the state or province arizona | 831 | 16-apr-2014 | 96.9 |
| greenwich is a park in the city london | 831 | 16-apr-2014 | 100.0 |

instances for many different relations



NELL: http://rtw.ml.cmu.edu/rtw/

# Example: Information Extraction

Recently-Learned Facts twitter Refresh

| instance | iteration | date learned | confidence |
|---|---|---|---|
| kelly_andrews is a female | 826 | 29-mar-2014 | 98.7 |
| investment_next_year is an economic sector | 829 | 10-apr-2014 | 95.3 |
| shibenik is a geopolitical entity that is an organization | 829 | 10-apr-2014 | 97.2 |
| quality_web_design_work is a character trait | 826 | 29-mar-2014 | 91.0 |
| mercedes_benz_cls_by_carlsson is an automobile manufacturer | 829 | 10-apr-2014 | 95.2 |
| social_work is an academic program at the university rutgers_university | 827 | 02-apr-2014 | 93.8 |
| dante wrote the book the_divine_comedy | 826 | 29-mar-2014 | 93.8 |
| willie_aames was born in the city los_angeles | 831 | 16-apr-2014 | 100.0 |
| kitt_peak is a mountain in the state or province arizona | 831 | 16-apr-2014 | 96.9 |
| greenwich is a park in the city london | 831 | 16-apr-2014 | 100.0 |

instances for many different relations

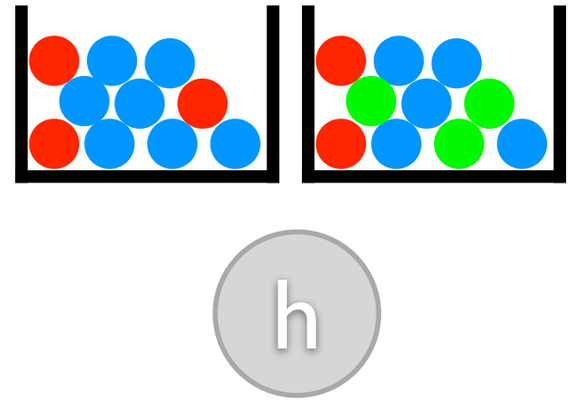
degree of certainty



NELL: http://rtw.ml.cmu.edu/rtw/

ProbLog by example:

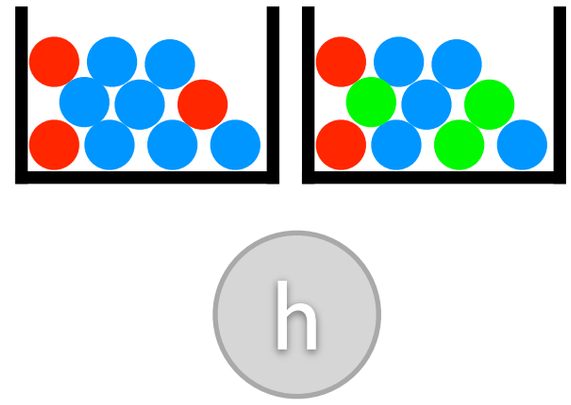
# A bit of gambling




- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

ProbLog by example:

# A bit of gambling



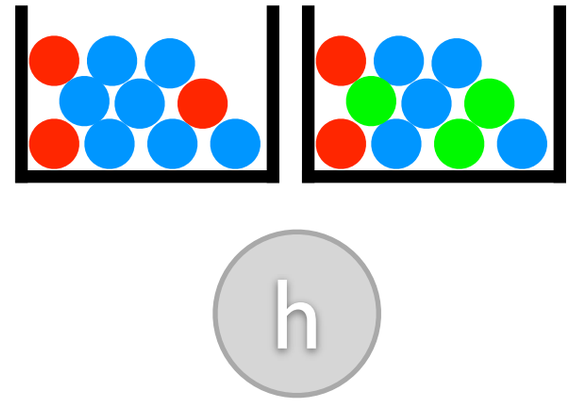

- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.
```

**probabilistic fact**: heads is true with probability 0.4 (and false with 0.6)

ProbLog by example:

# A bit of gambling



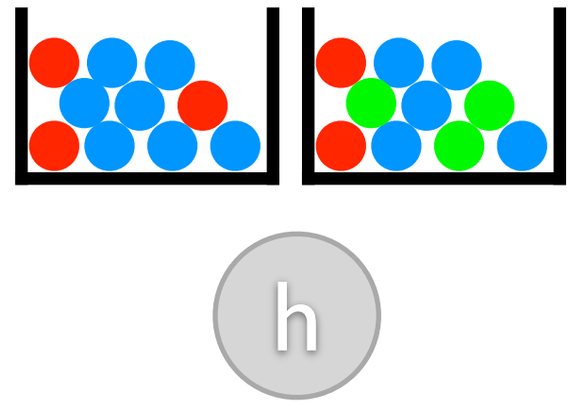

- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
```

**annotated disjunction**: first ball is red with probability 0.3 and blue with 0.7

ProbLog by example:

# A bit of gambling



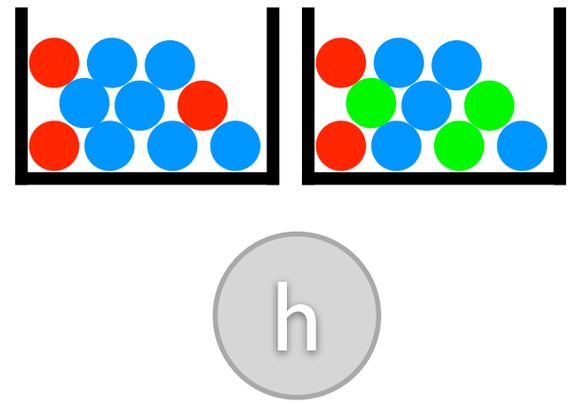

- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green);
                   0.5 :: col(2,blue).
```

**annotated disjunction**: second ball is red with probability 0.2, green with 0.3, and blue with 0.5

ProbLog by example:

# A bit of gambling



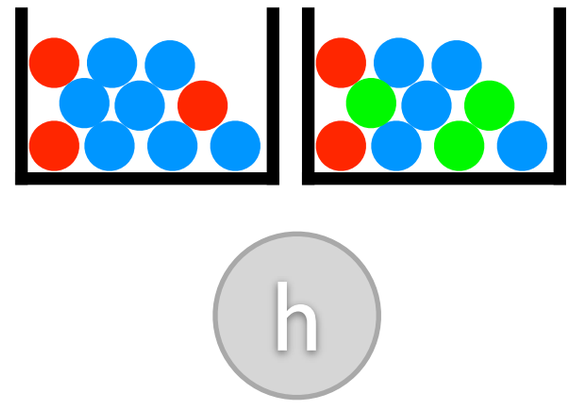

- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green);
                   0.5 :: col(2,blue).

win :- heads, col(_,red).
```

**logical rule** encoding background knowledge

ProbLog by example:

# A bit of gambling



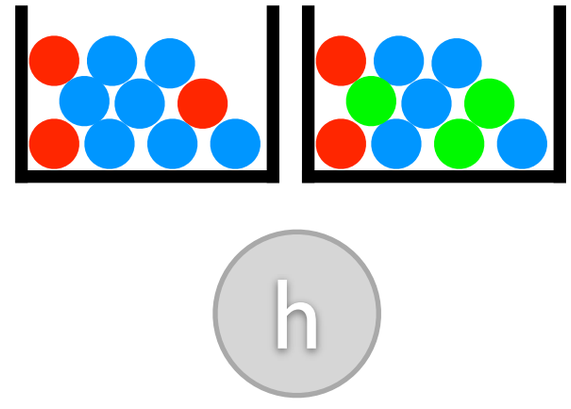

- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green);
                   0.5 :: col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

**logical rule** encoding background knowledge

ProbLog by example:

# A bit of gambling




- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green);
                   0.5 :: col(2,blue).
```

**probabilistic choices**

```
win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

**consequences**

# Questions

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

- Probability of **`win`**?
- Probability of **`win`** given **`col(2,green)`**?
- Most probable world where **`win`** is true?

# Questions

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

**marginal probability**

- Probability of `win`?

**query**

- Probability of `win` given `col(2,green)`?
- Most probable world where `win` is true?

# Questions

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

**marginal probability**

- Probability of **`win`**?

**conditional probability**

- Probability of **`win`** given **`col(2,green)`**?

**evidence**

- Most probable world where **`win`** is true?

# Questions

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

**marginal probability**

- Probability of **`win`**?

**conditional probability**

- Probability of **`win`** given **`col(2,green)`**?

- Most probable world where **`win`** is true?

**MPE inference**

# Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col( ,red).
win :- col(1,C), col(2,C).
```

# Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col( ,red).
win :- col(1,C), col(2,C).
```

# Possible Worlds

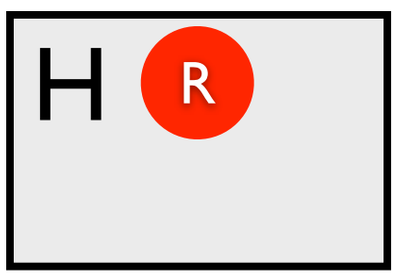
```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

0.4

# Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

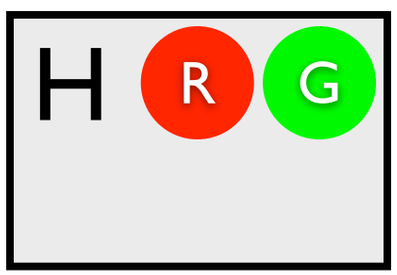
0.4 ×0.3



# Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

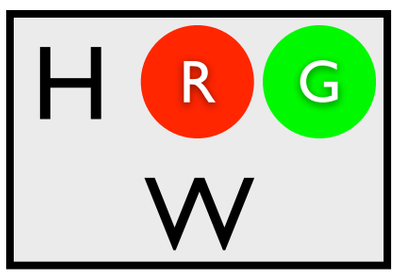
0.4 ×0.3 ×0.3



# Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

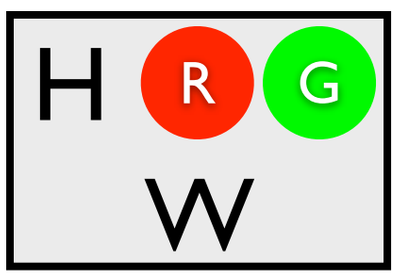
0.4 ×0.3 ×0.3



# Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col( ,red).
win :- col(1,C), col(2,C).
```

$0.4 \times 0.3 \times 0.3$

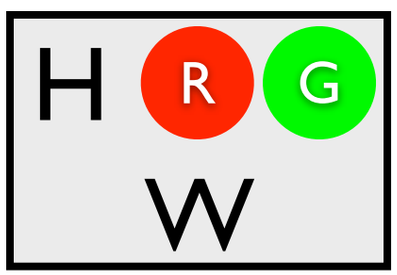



$(1-0.4)$

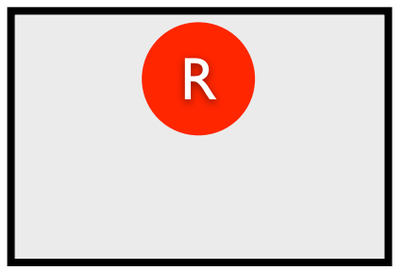
# Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col( ,red).
win :- col(1,C), col(2,C).
```

$0.4 \times 0.3 \times 0.3$



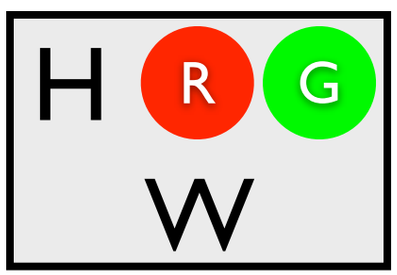

$(1-0.4) \times 0.3$



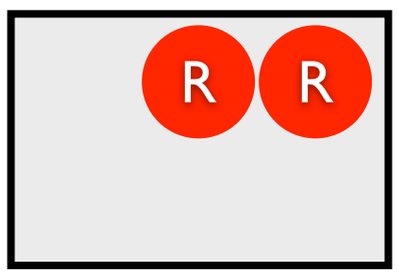
# Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

0.4 ×0.3 ×0.3



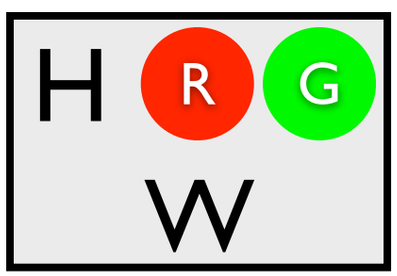

(1−0.4)×0.3 ×0.2



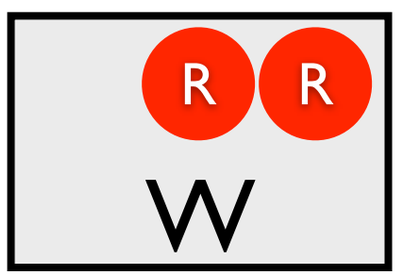
# Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

$0.4 \times 0.3 \times 0.3$



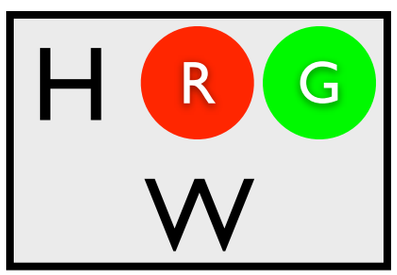

$(1-0.4) \times 0.3 \times 0.2$



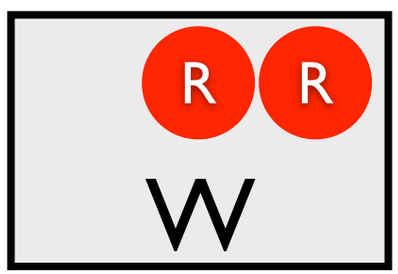
# Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

$0.4 \times 0.3 \times 0.3$




$(1-0.4) \times 0.3 \times 0.2$

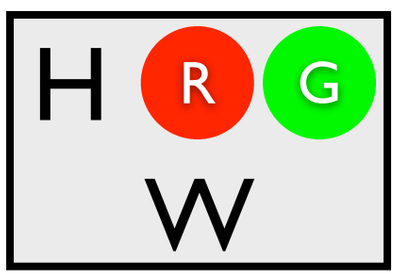



$(1-0.4)$

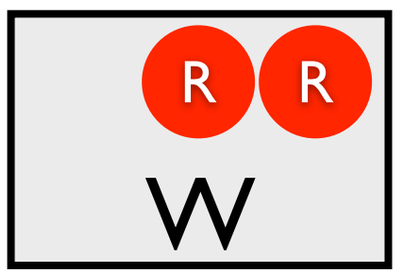
# Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col( ,red).
win :- col(1,C), col(2,C).
```

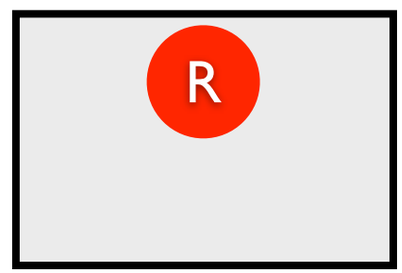
0.4 ×0.3 ×0.3




(1−0.4)×0.3 ×0.2



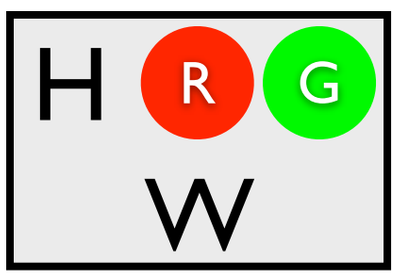

(1−0.4)×0.3



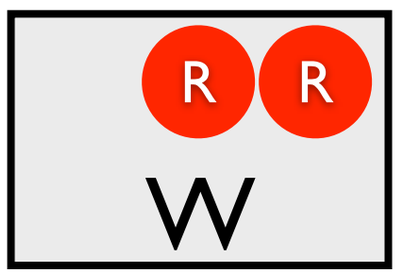
# Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

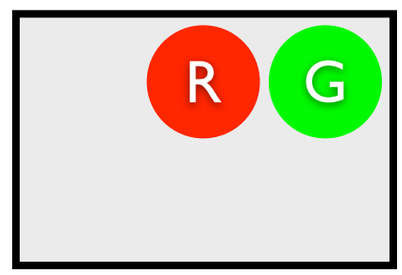
0.4 ×0.3 ×0.3




(1−0.4)×0.3 ×0.2



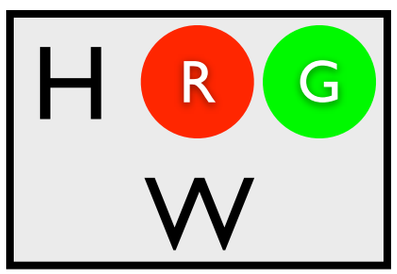

(1−0.4)×0.3 ×0.3



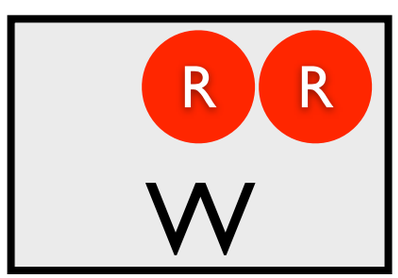
# Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

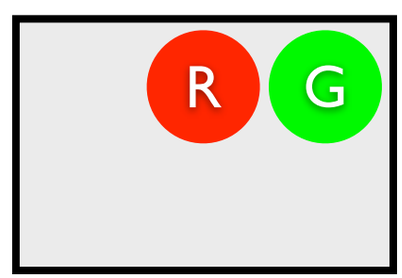
$0.4 \times 0.3 \times 0.3$




$(1-0.4) \times 0.3 \times 0.2$



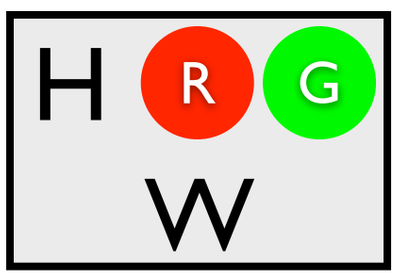

$(1-0.4) \times 0.3 \times 0.3$



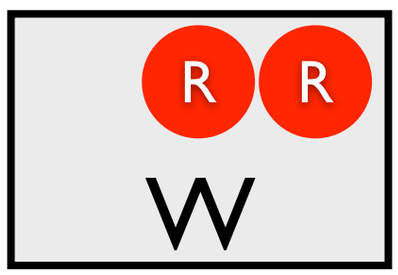
# Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col( ,red).
win :- col(1,C), col(2,C).
```

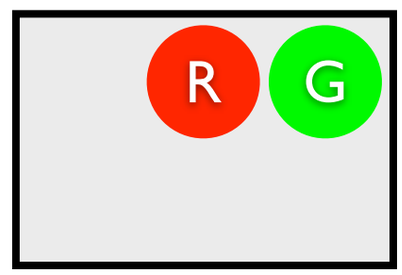
0.4 ×0.3 ×0.3




(1−0.4)×0.3 ×0.2

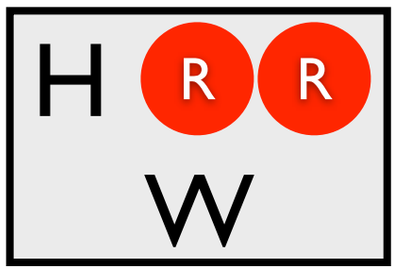



(1−0.4)×0.3 ×0.3

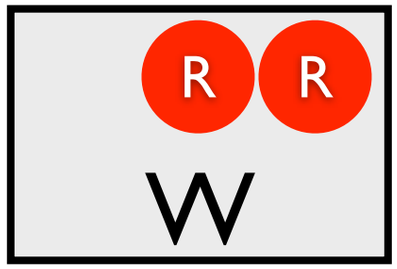


# All Possible Worlds

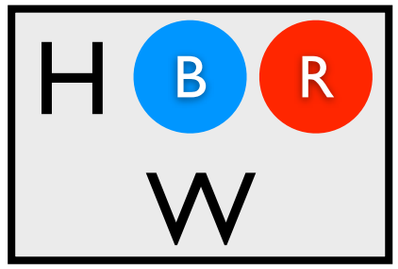
0.024



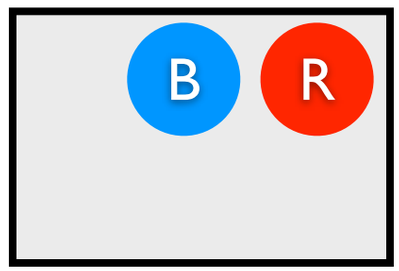

0.036



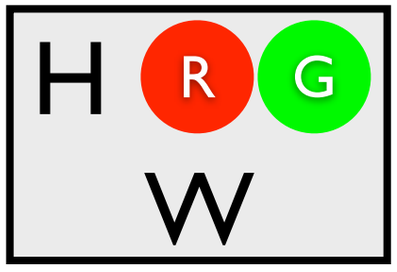

0.056



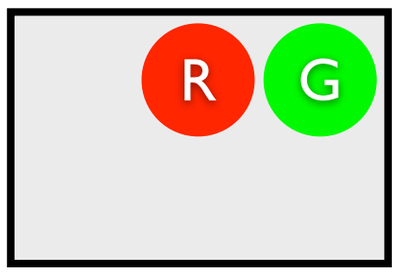

0.084



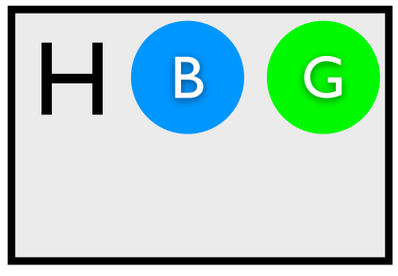

0.036



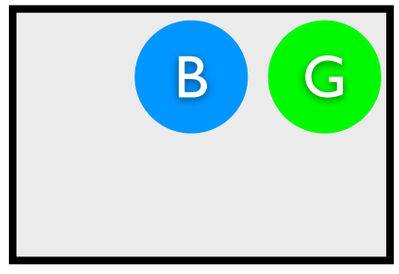

0.054



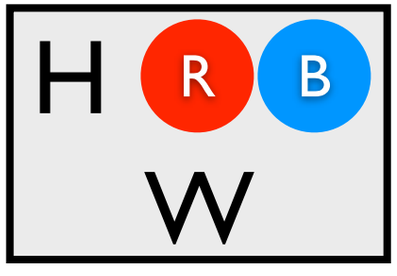

0.084



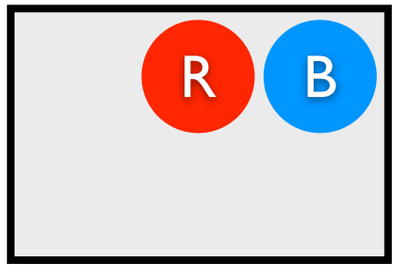

0.126



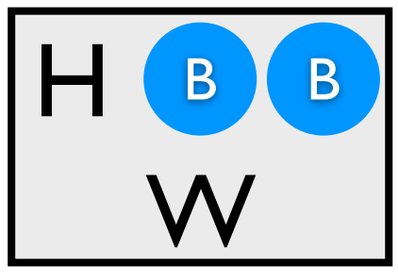

0.060



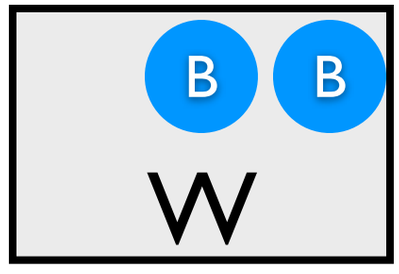

0.090




0.140

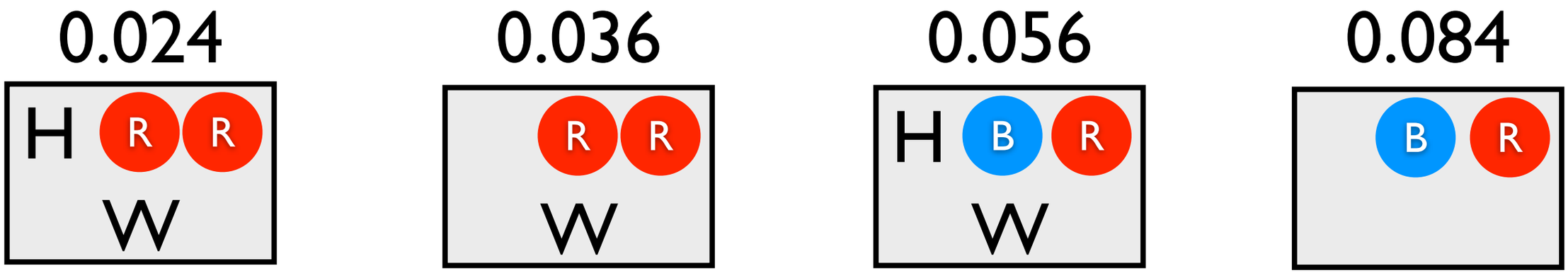



0.210



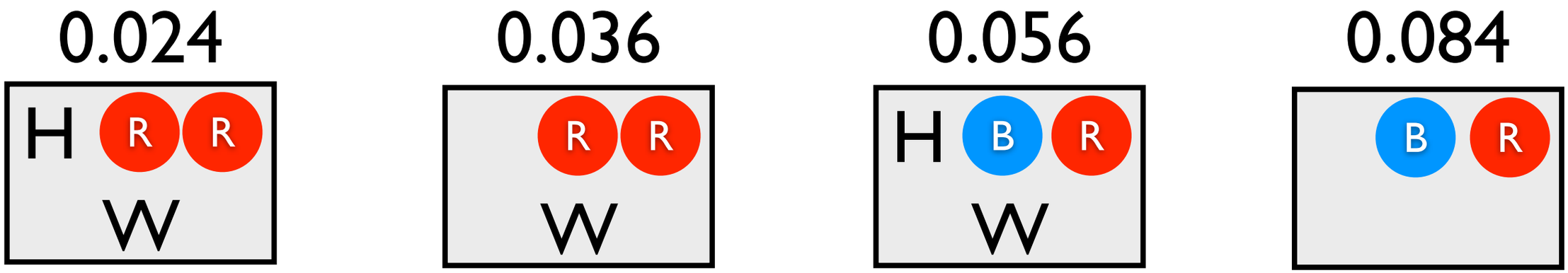
# Most likely world where **`win`** is true?

MPE Inference



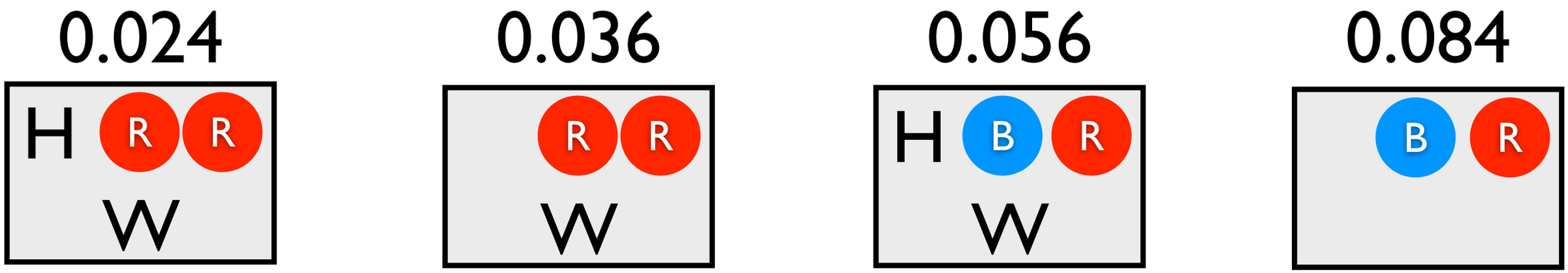
# Most likely world where **`win`** is true?

MPE Inference



# Most likely world where `col(2,blue)` is false?

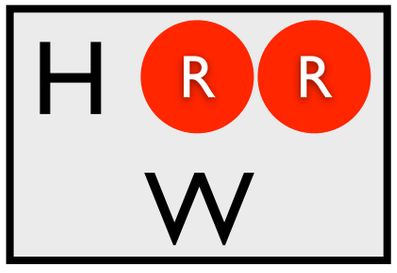
MPE Inference



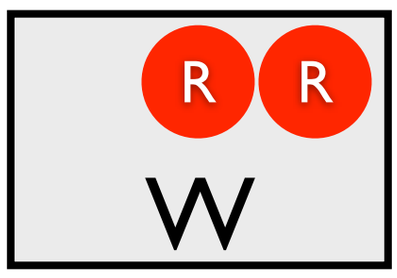
# Most likely world where `col(2,blue)` is false?

MPE Inference

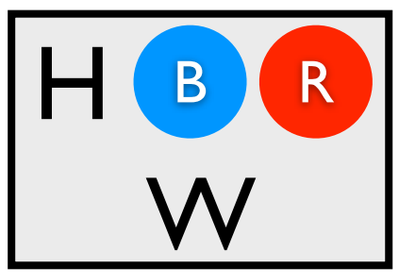
0.024



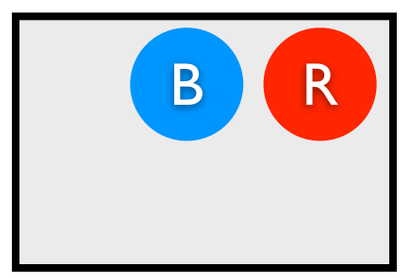

0.036



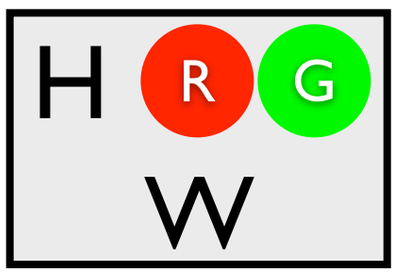

0.056



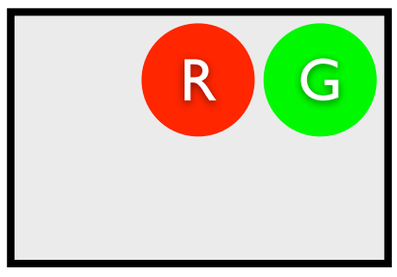

0.084



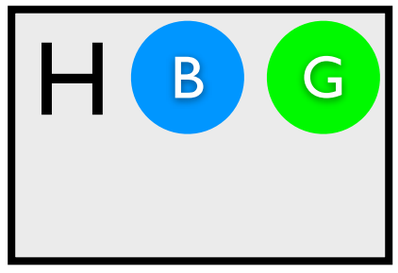

0.036



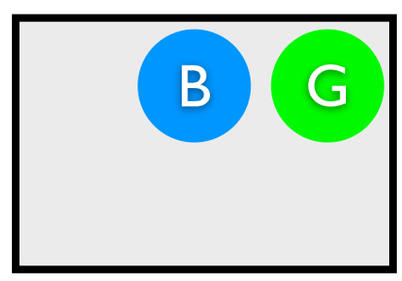

0.054



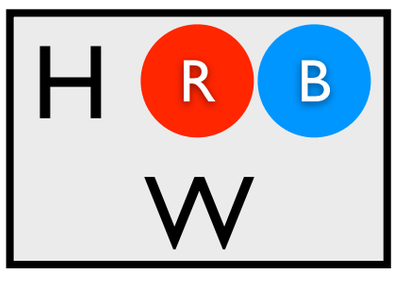

0.084



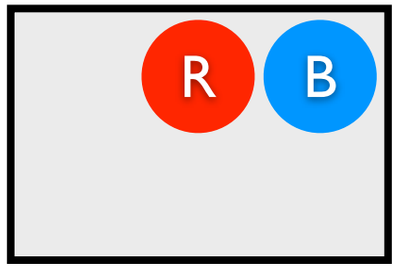

0.126



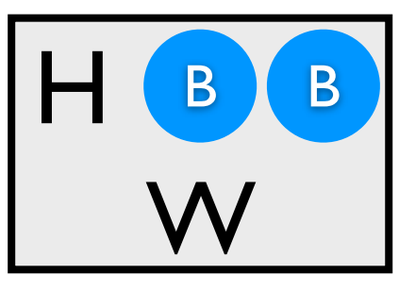

0.060



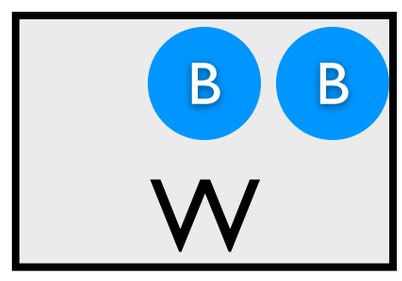

0.090




0.140

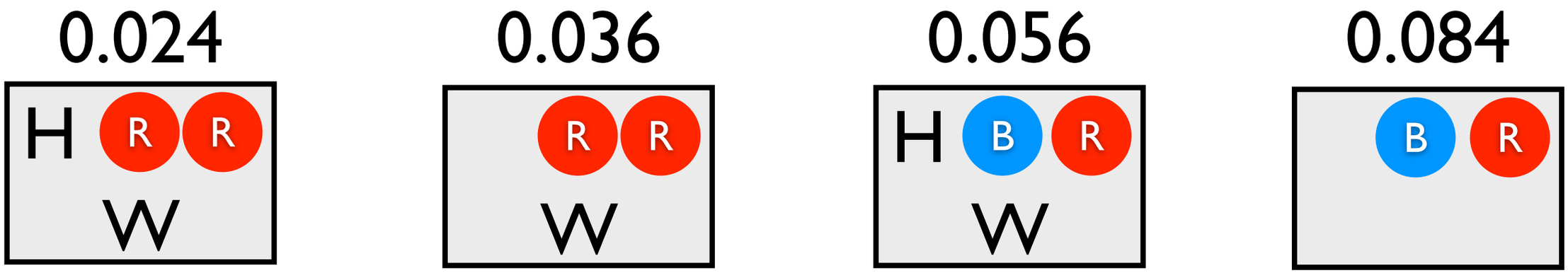



0.210



# P(win)=?

Marginal Probability

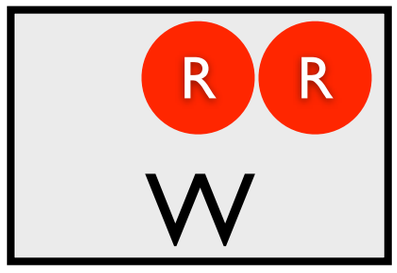


# P(win)= Σ

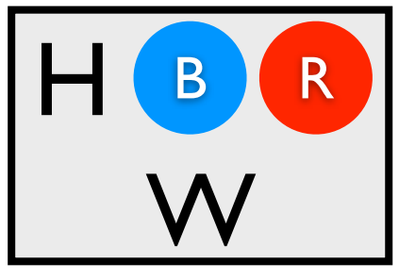
Marginal Probability

0.024

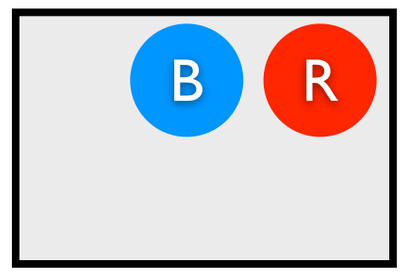

0.036



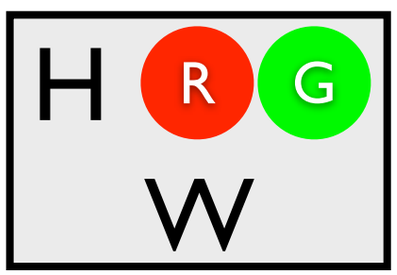

0.056



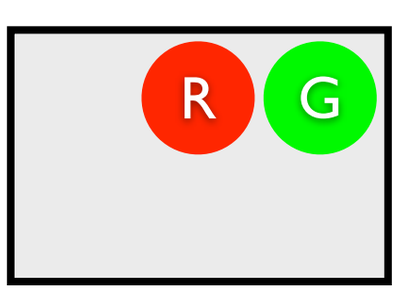

0.084



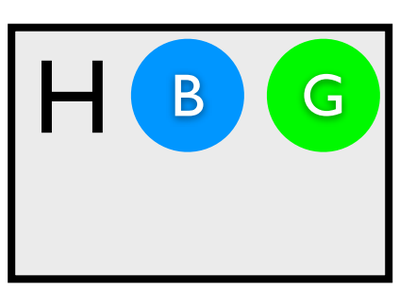

0.036



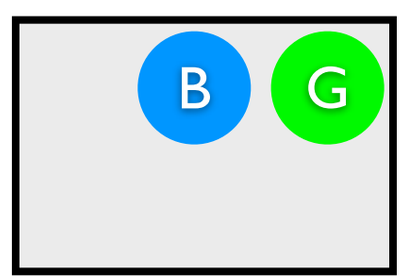

0.054



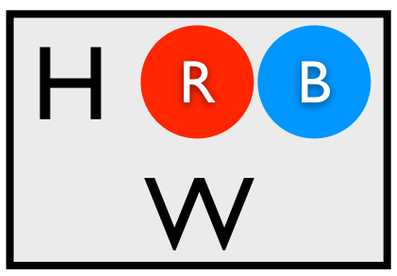

0.084



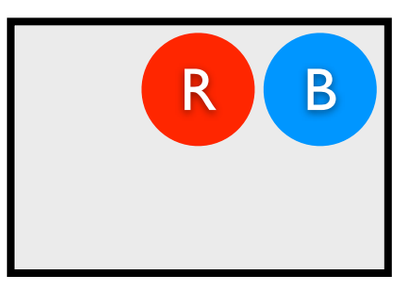

0.126



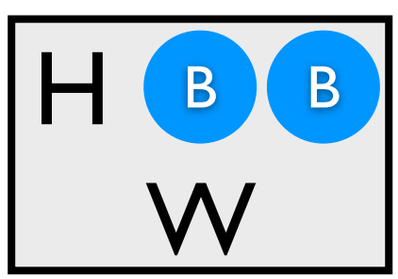

0.060



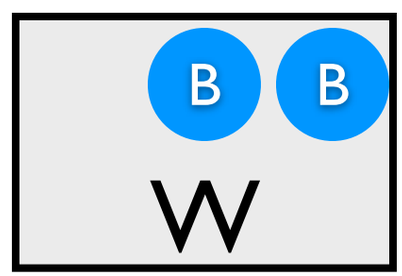

0.090




0.140




0.210

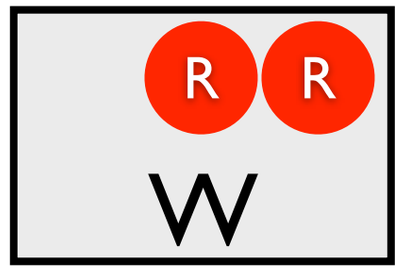


# P(win)= Σ =0.562

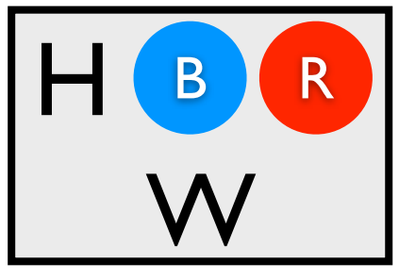
Marginal Probability

0.024

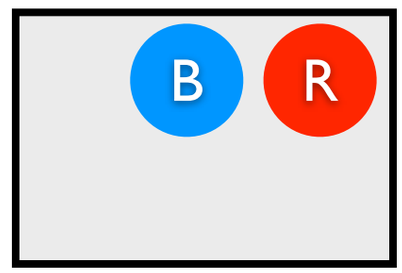

0.036



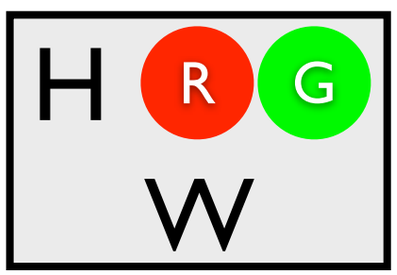

0.056



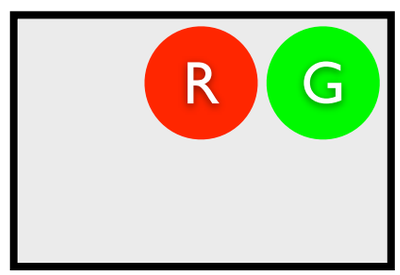

0.084



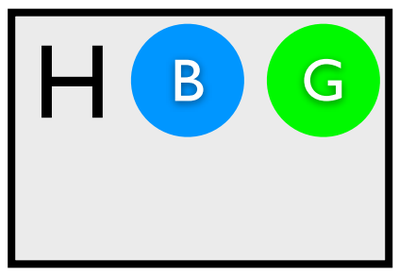

0.036



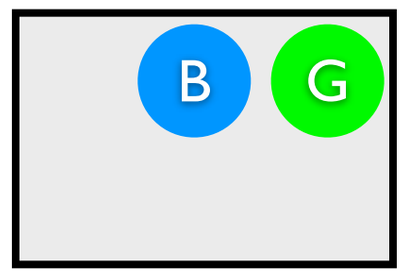

0.054



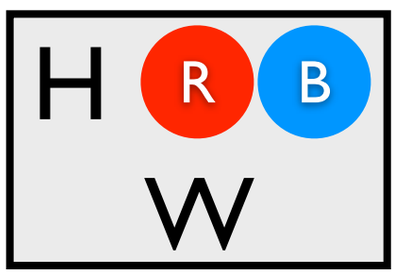

0.084



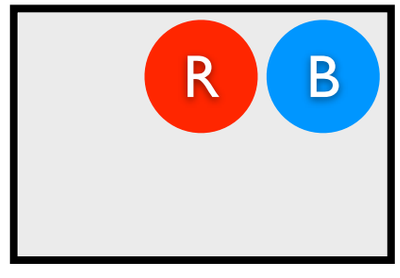

0.126



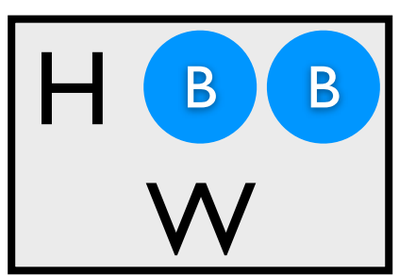

0.060



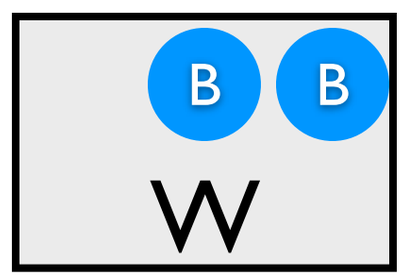

0.090




0.140



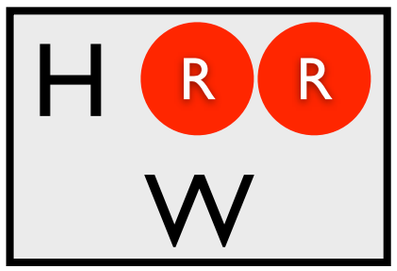

0.210



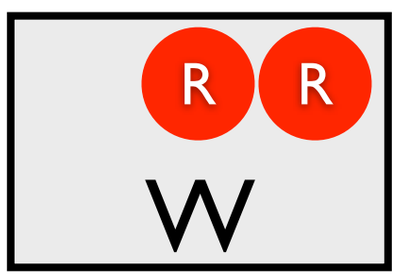
P(win|col(2,green))= ?

Conditional Probability

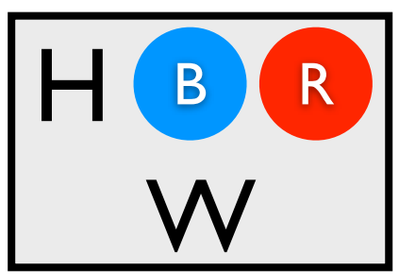
0.024



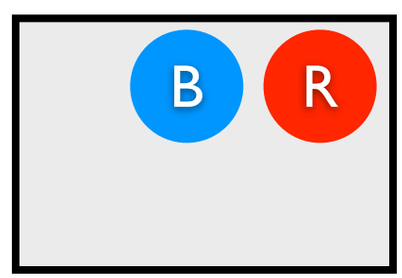

0.036



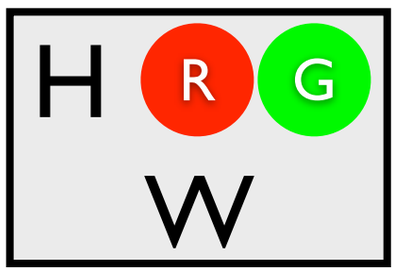

0.056



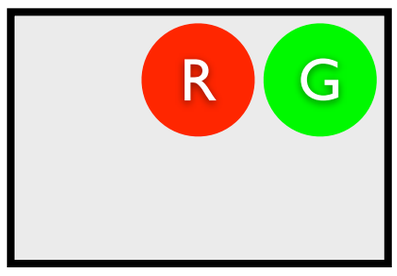

0.084



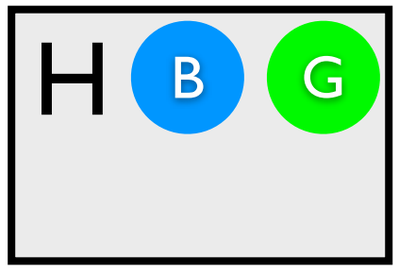

0.036



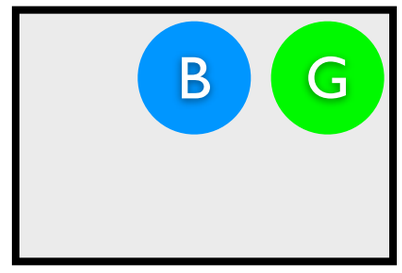

0.054



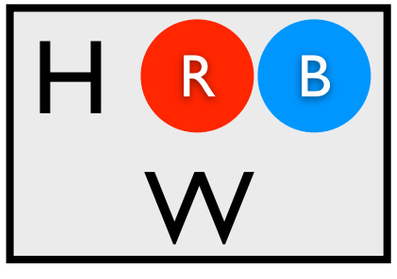

0.084



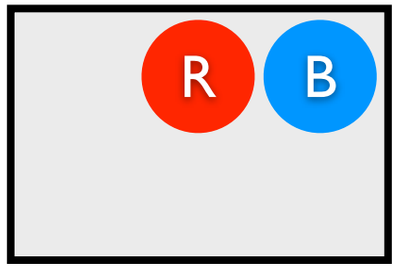

0.126



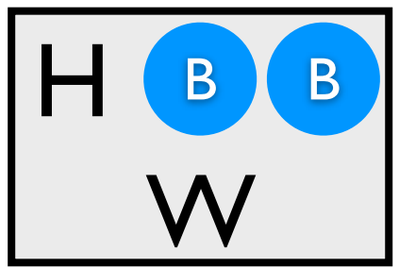

0.060



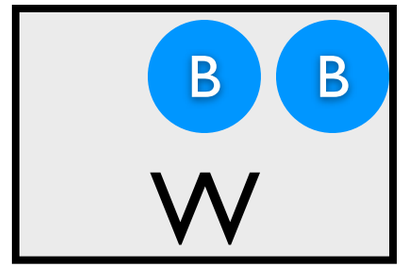

0.090




0.140




0.210

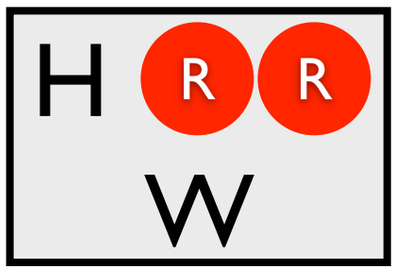


$$P(win|col(2,green)) = \Sigma/\Sigma$$

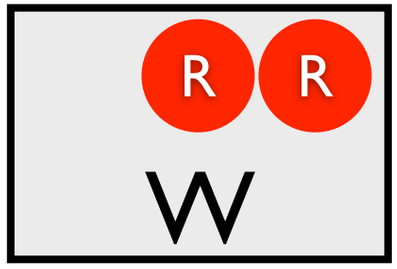
$$= P(win \wedge col(2,green))/P(col(2,green))$$

Conditional Probability

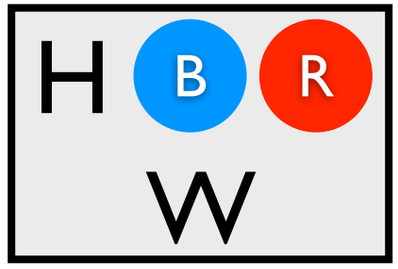
0.024



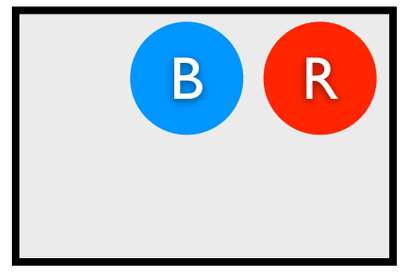

0.036



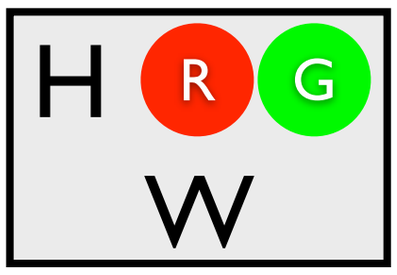

0.056



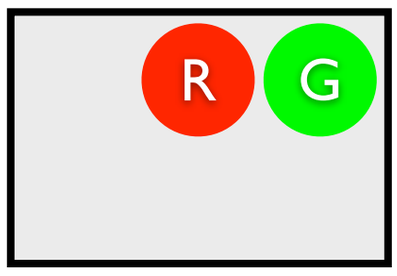

0.084



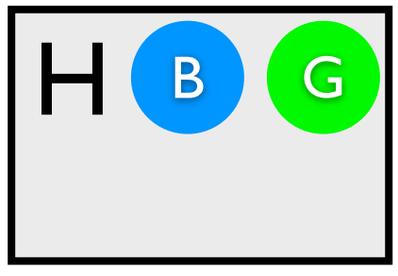

0.036



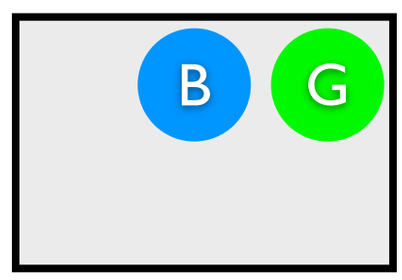

0.054



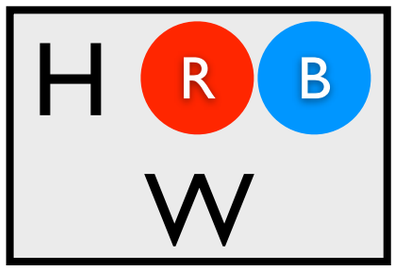

0.084



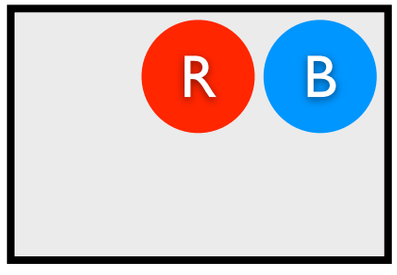

0.126



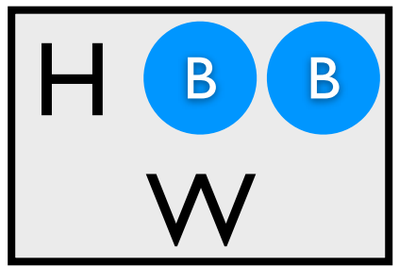

0.060



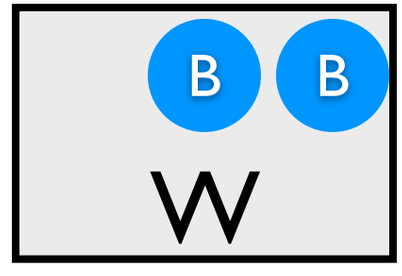

0.090




0.140



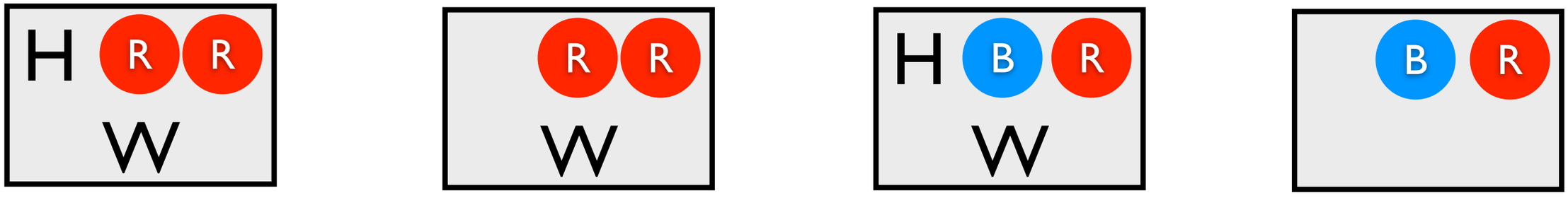

0.210

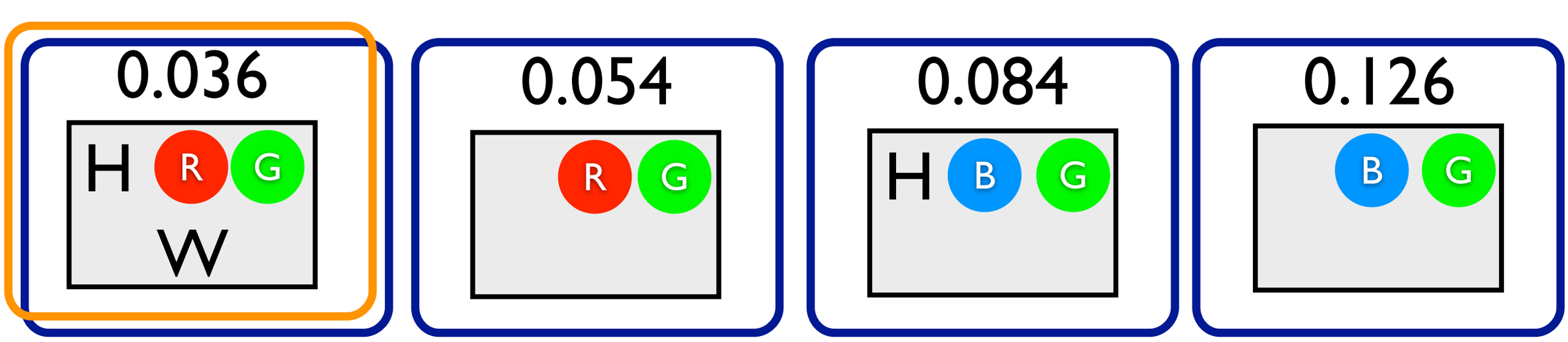


$$P(win|col(2,green)) = \Sigma/\Sigma$$

$$= P(win \wedge col(2,green))/P(col(2,green))$$

Conditional Probability

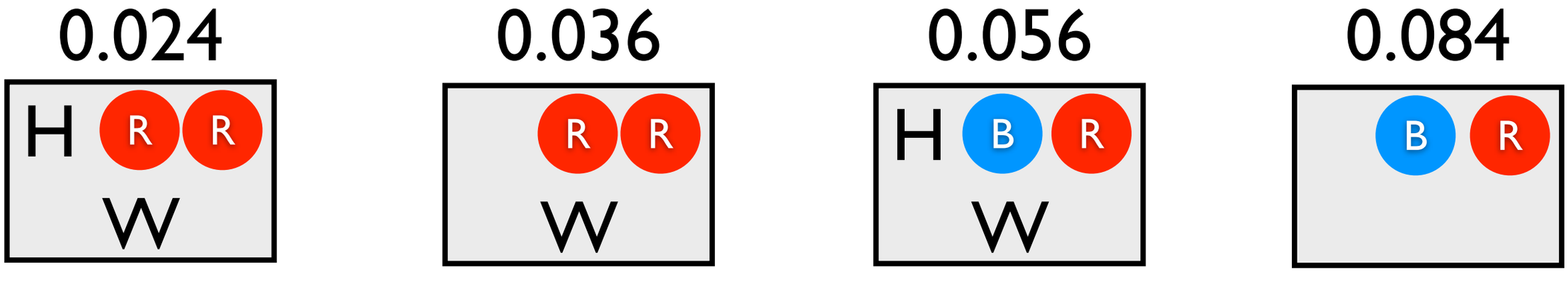


$$P(win|col(2,green)) = \Sigma/\Sigma$$

$$= 0.036/0.3 = 0.12$$

Conditional Probability



# Inference in PLP

- As in Prolog and logic programming
  - proof-based
- As in Answer Set Programming
  - model based
- As in Probabilistic Programming
  - sampling

# Parameter Learning

e.g., webpage classification model

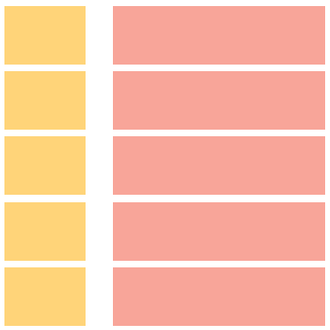
for each *CLASS1*, *CLASS2* and each *WORD*

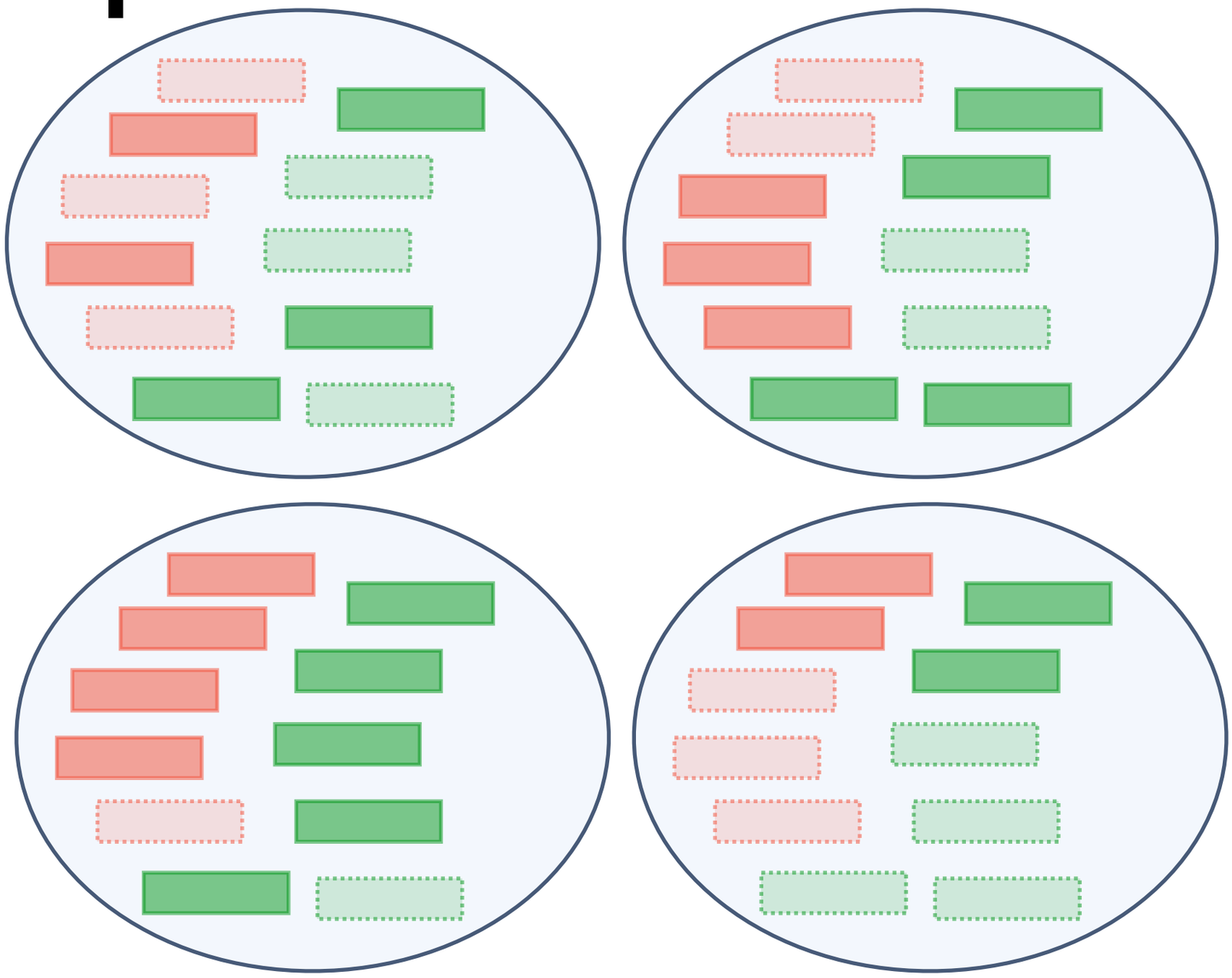
?? :: link_class(Source,Target,*CLASS1*,*CLASS2*).
?? :: word_class(*WORD*,*CLASS*).

```
class(Page,C)    :-  has_word(Page,W), word_class(W,C).

class(Page,C)        :-  links_to(OtherPage,Page),
    class(OtherPage,OtherClass),
                         link_class(OtherPage,Page,OtherClass,C).
```

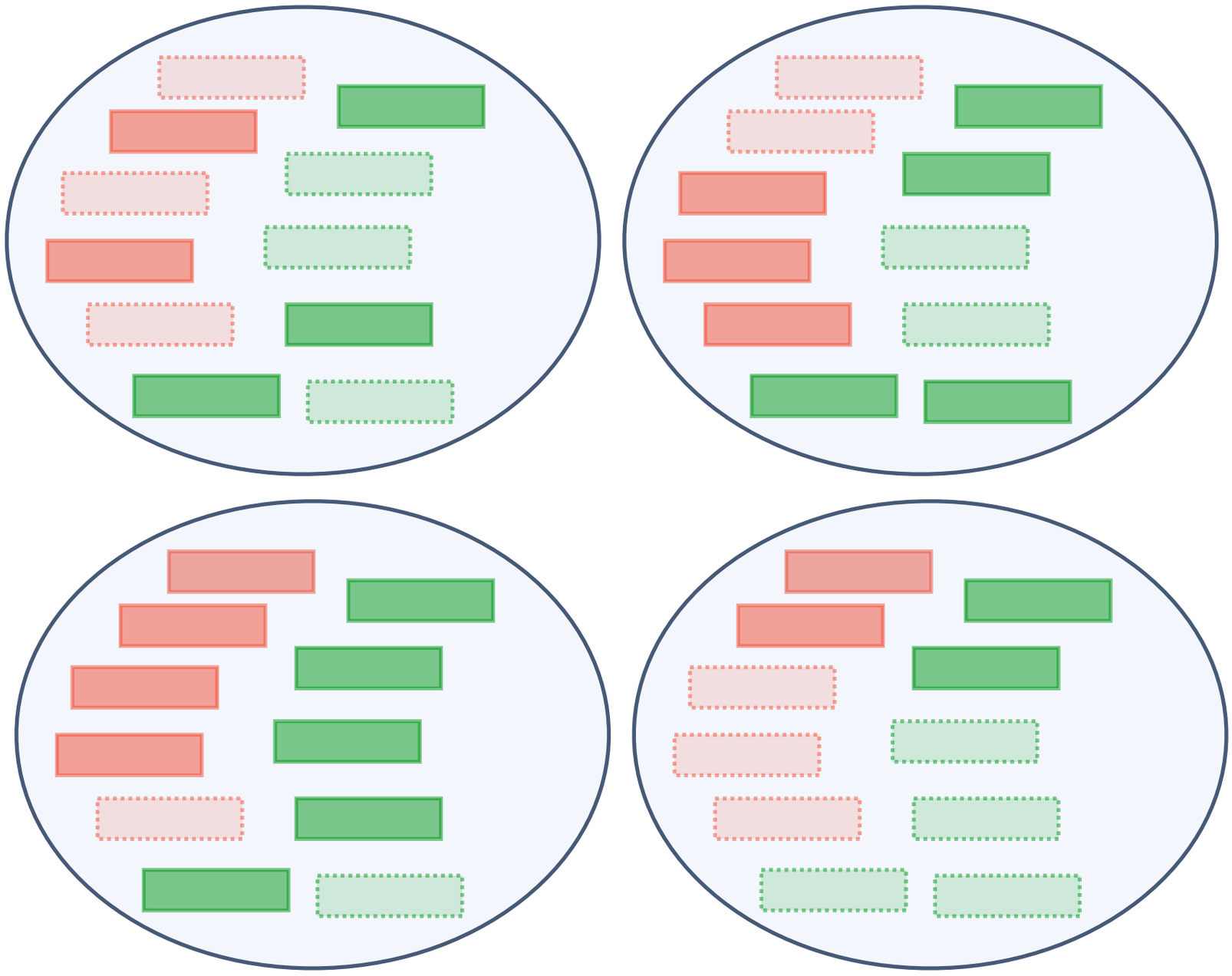
# Sampling Interpretations

# Sampling Interpretations

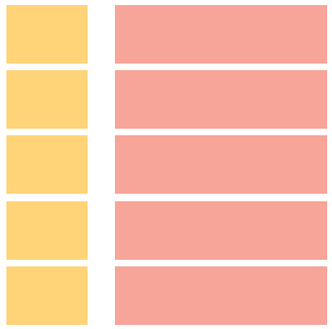


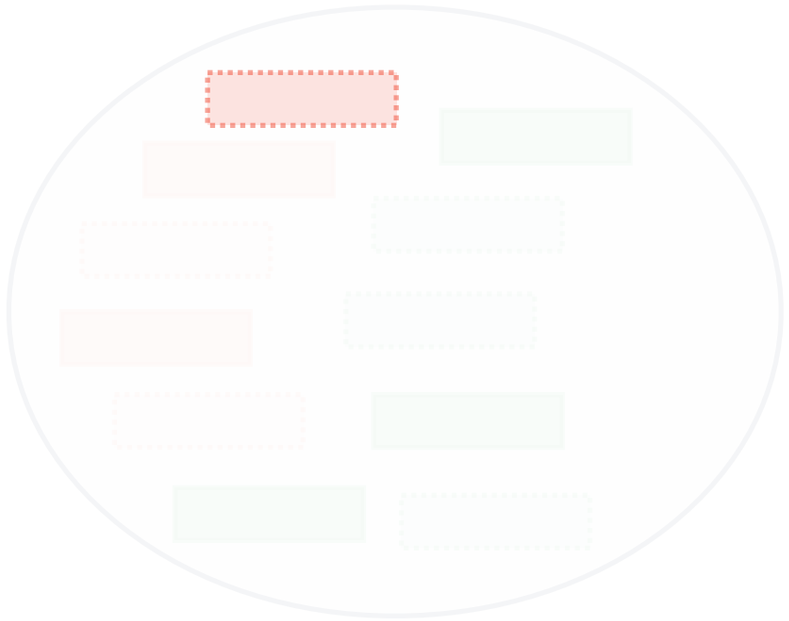


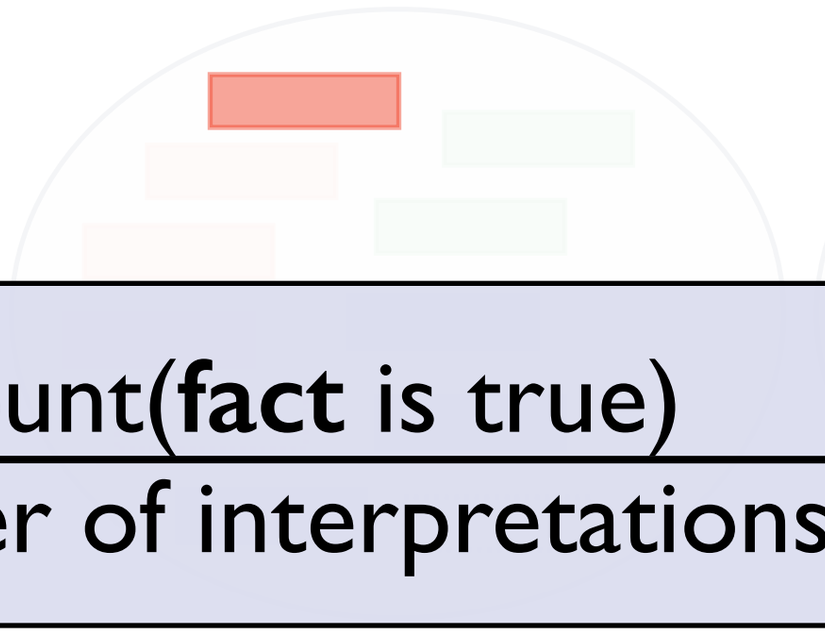
# Parameter Estimation



# Parameter Estimation

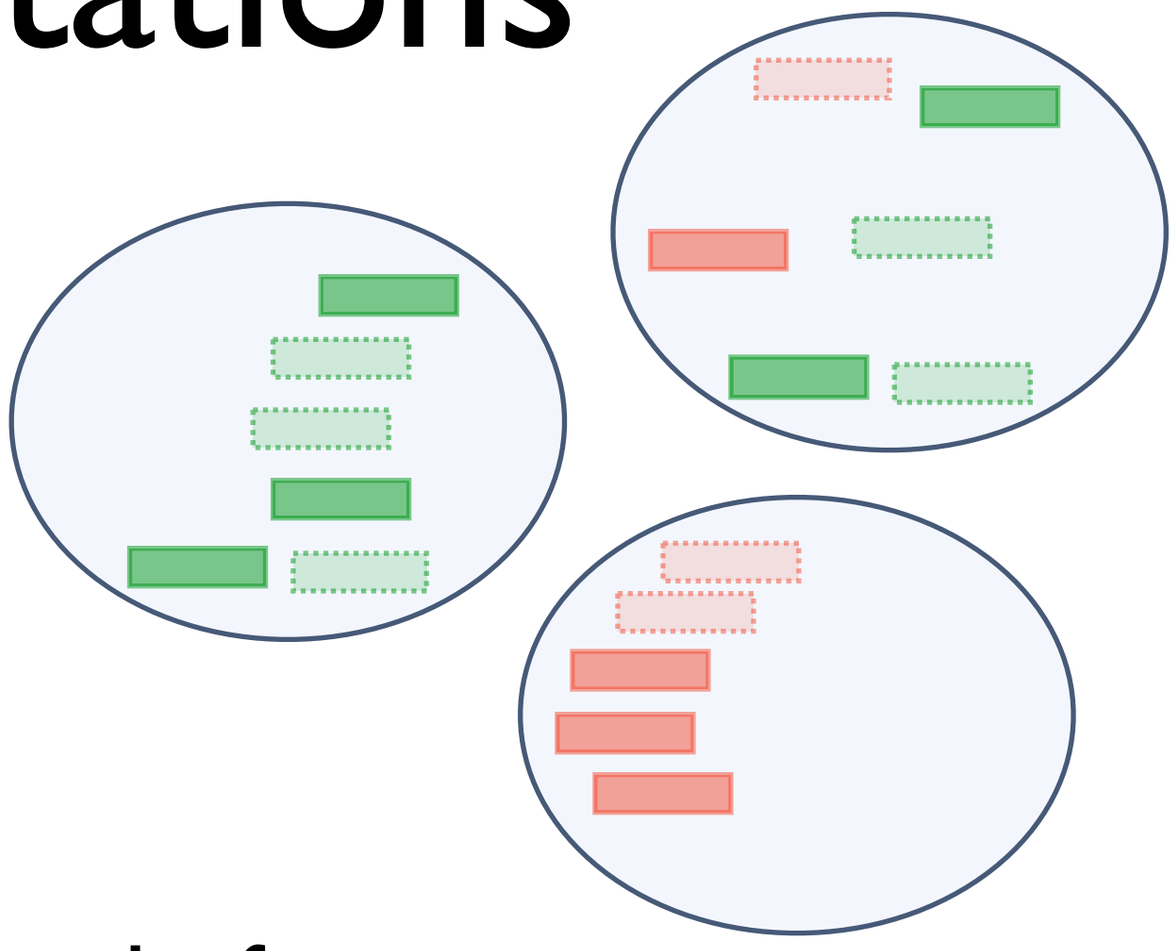






$$p(\textbf{fact}) = \frac{\text{count}(\textbf{fact} \text{ is true})}{\text{Number of interpretations}}$$

# Learning from partial interpretations



- Not all facts observed
- Soft-EM
- use **expected count** instead of **count**
- **P(Q |E) -- conditional queries !**



[Gutmann et al, ECML 11; Fierens et al, TPLP 14]

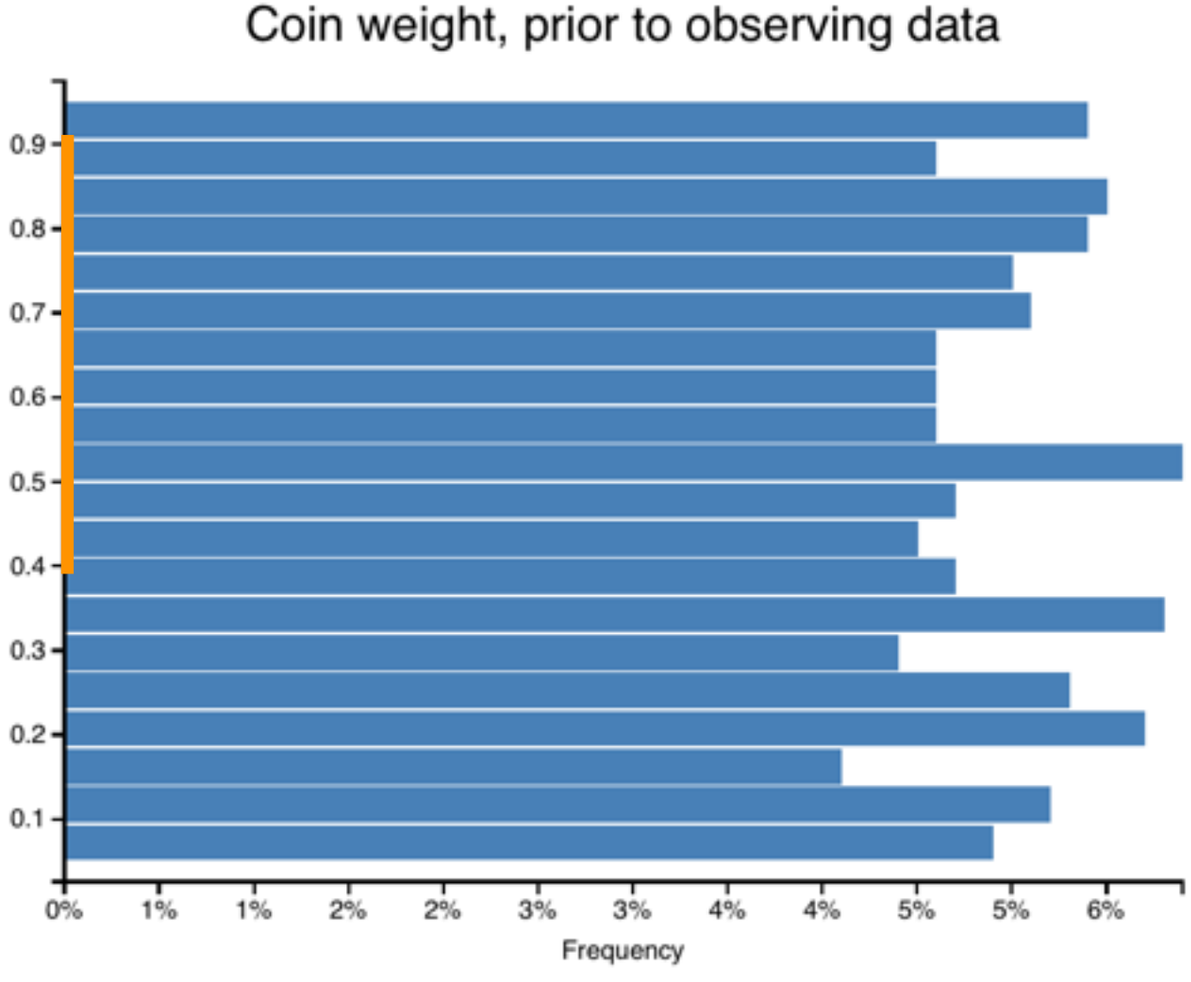
# Bayesian Parameter Learning

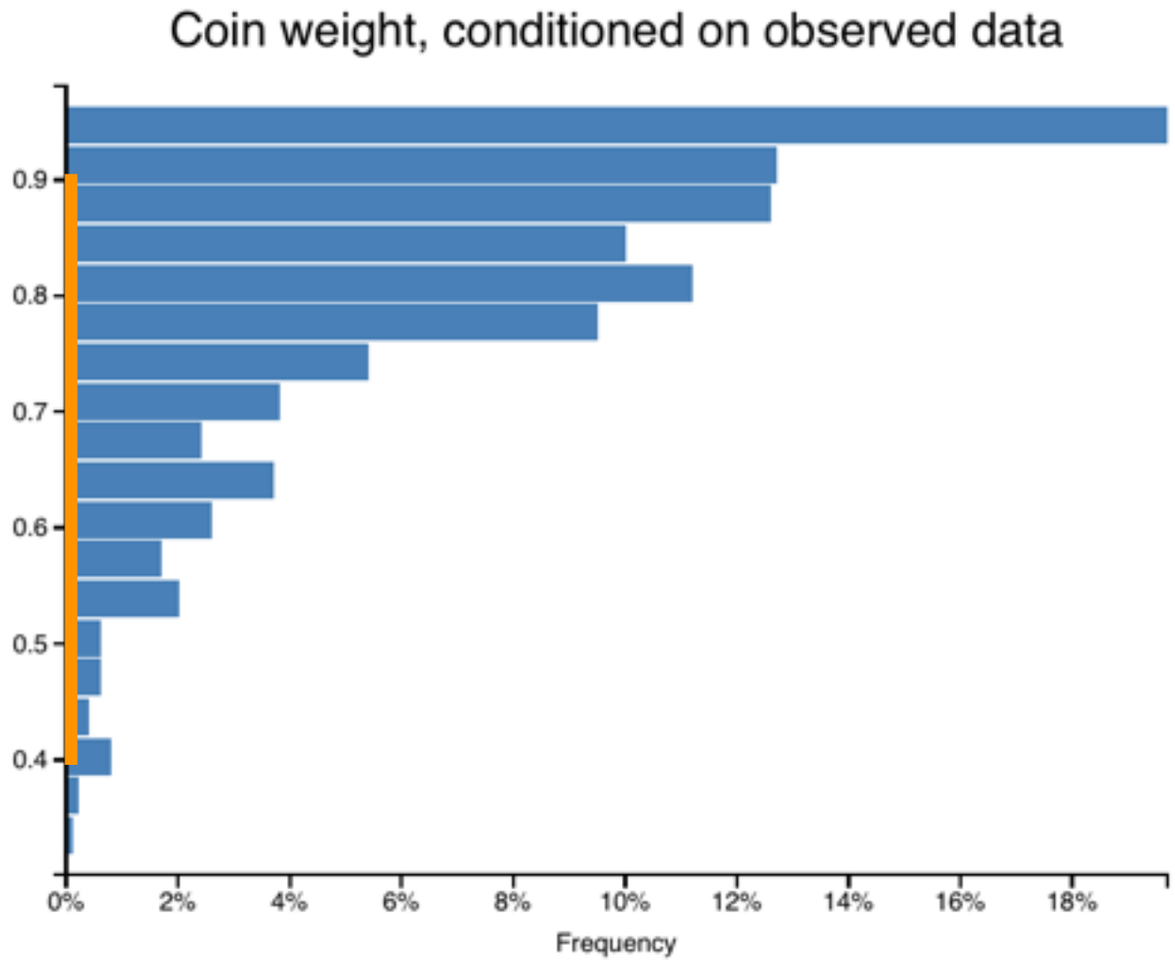
- Learning as inference (e.g., Church)
- Prior distributions for parameters
- Given data, find most likely parameter values

# Example

[from probmods.org]

- Flipping a coin with unknown weight
- Prior: uniform distribution on [0,1]
- Observation: 5x heads in a row
- Sampling from Church model:



# ProbLog Example

**prior**

```
0.05::weight(C,0.1); 0.2::weight(C,0.3); 0.5::weight(C,0.5);
                  0.2::weight(C,0.7); 0.05::weight(C,0.9) :- coin(C).
```

```
Param::toss(_,Param,_).
heads(C,R) :- weight(C,Param),toss(C,Param,R).
tails(C,R) :- weight(C,Param),\+toss(C,Param,R).

data(C,[]).
data(C,[h|R]) :- heads(C,R), data(C,R).
data(C,[t|R]) :- tails(C,R), data(C,R).
```

```
coin(c1).
coin(c2).
param(0.1).
param(0.3).
param(0.5).
param(0.7).
param(0.9).
```

```
query(weight(C,X)) :- coin(C),param(X).
```

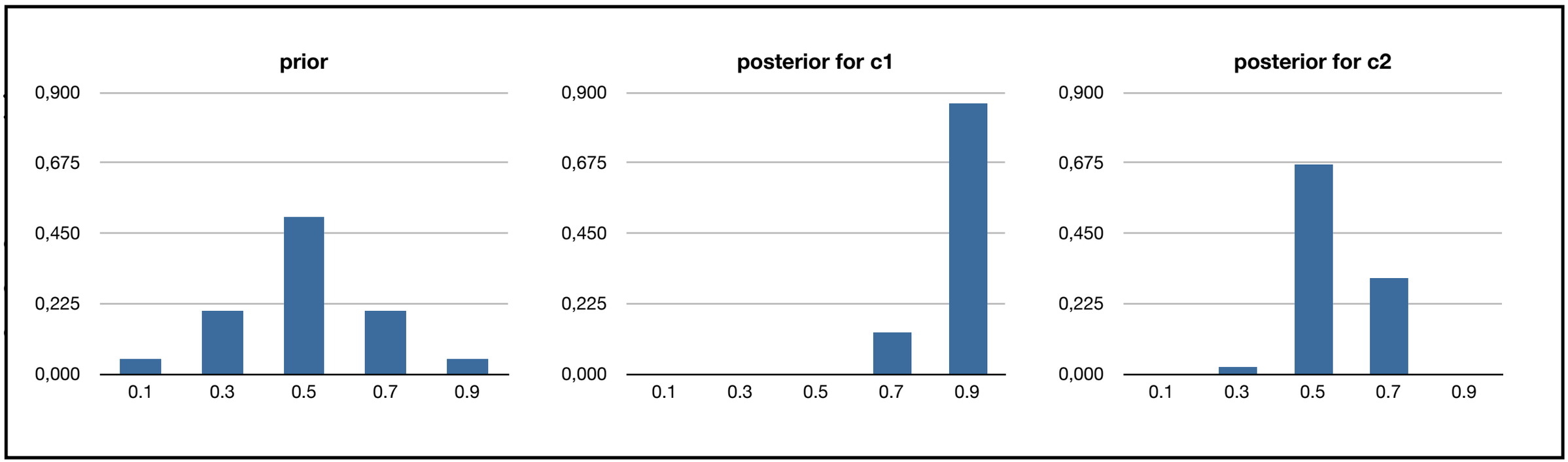
**ask for posterior**

```
evidence(data(c1,[h,h,h,h,h,h,h,h,h,h,h,h,h,h]),true).
evidence(data(c2,[h,t,h,h,h,h,h,t,t,h,t,t,h]),true).
```

**data**

# ProbLog Example

```
0.05::weight(C,0.1); 0.2::weight(C,0.3); 0.5::weight(C,0.5);
                    0.2::weight(C,0.7); 0.05::weight(C,0.9) :- coin(C).
```



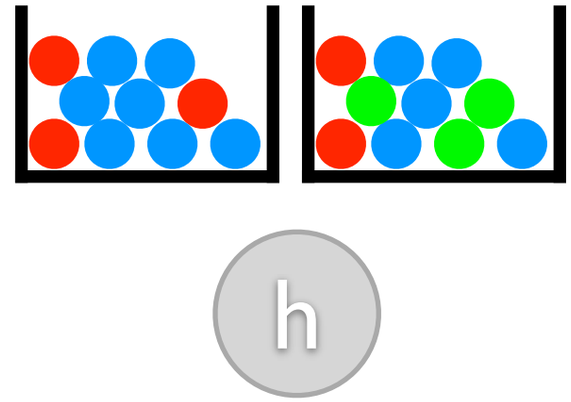

```
query(weight(C,X)) :- coin(C),param(X).

evidence(data(c1,[h,h,h,h,h,h,h,h,h,h,h,h,h,h]),true).
evidence(data(c2,[h,t,h,h,h,h,h,t,t,h,t,t,h]),true).
```

# Some Probabilistic Programming Languages outside LP

- IBAL [Pfeffer 01]
- Figaro [Pfeffer 09]
- Church [Goodman et al 08 ]
- BLOG [Milch et al 05]
- Venture [Mansingha et al.]
- Anglican and Probabilistic-C [Wood et al].
- and many more appearing recently

Church by example:

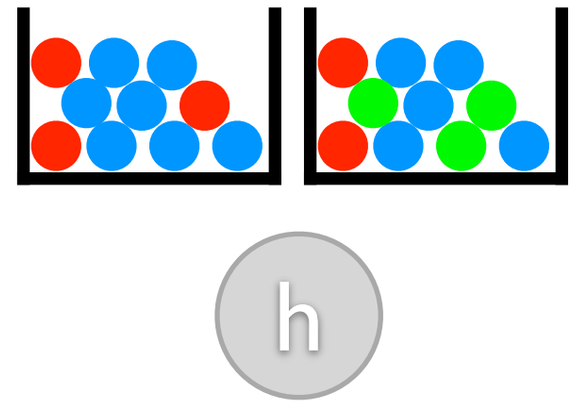
# A bit of gambling




- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

Church by example:

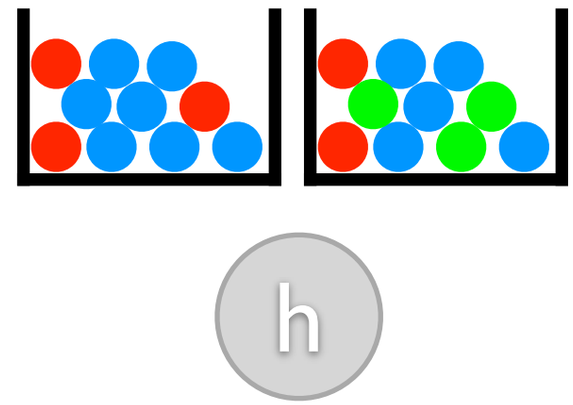
# A bit of gambling




- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

Church by example:

# A bit of gambling

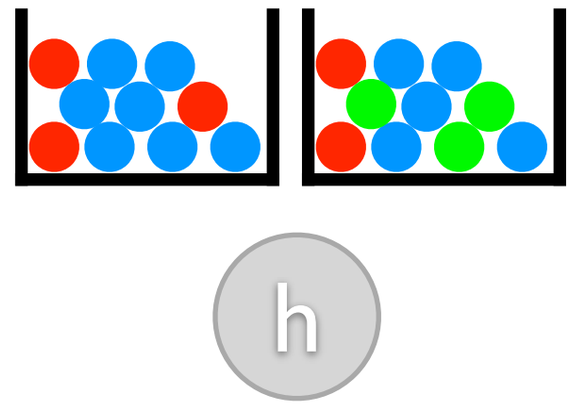



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))
```

Church by example:

# A bit of gambling

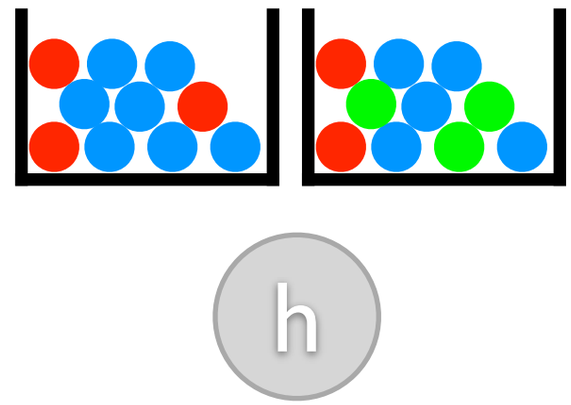



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))
```

Church by example:

# A bit of gambling

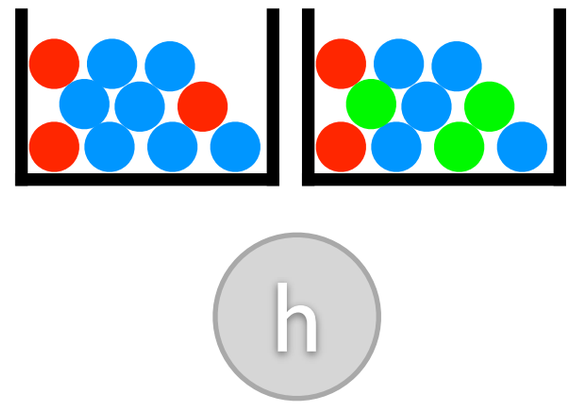



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))
```

Church by example:

# A bit of gambling

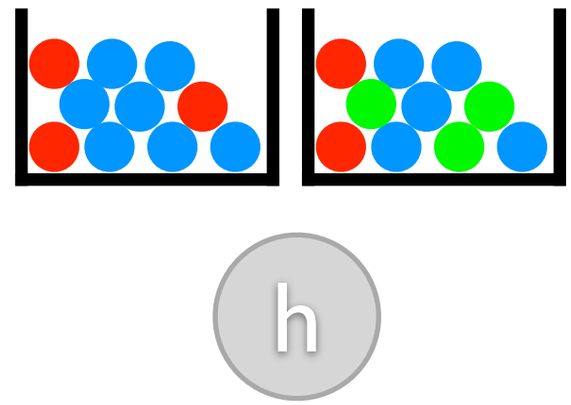



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                  (multinomial '(red green blue) '(0.2 0.3 0.5)))))
```

Church by example:

# A bit of gambling

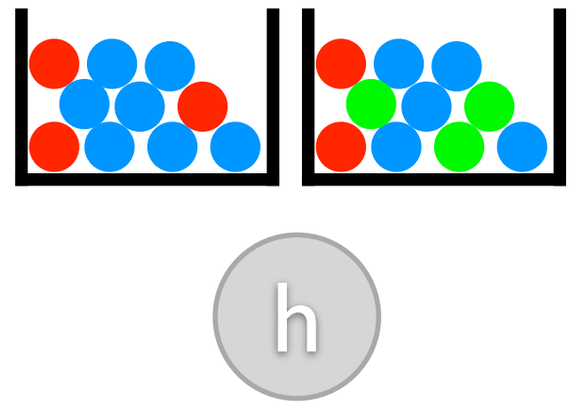



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                  (multinomial '(red green blue) '(0.2 0.3 0.5)))))
```

Church by example:

# A bit of gambling

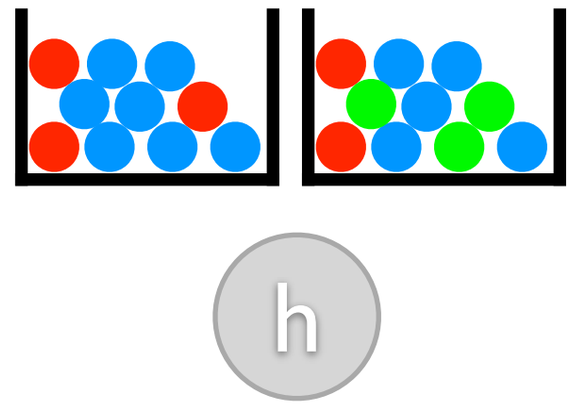



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                      (multinomial '(red green blue) '(0.2 0.3 0.5)))))

(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))
```

Church by example:

# A bit of gambling

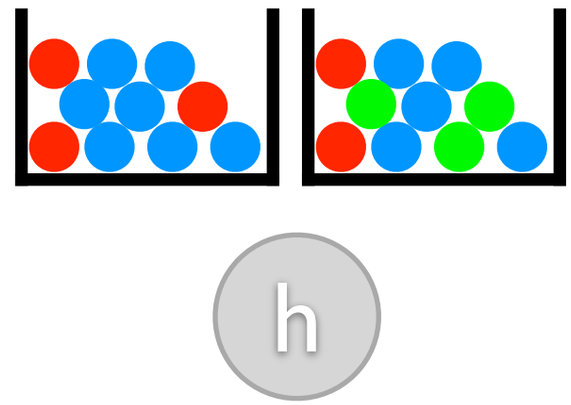



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                      (multinomial '(red green blue) '(0.2 0.3 0.5)))))

(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))

(define win1 (and (heads) redball))
```

Church by example:

# A bit of gambling

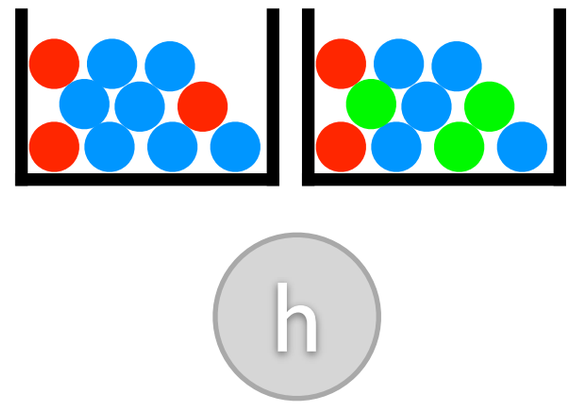



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                      (multinomial '(red green blue) '(0.2 0.3 0.5)))))

(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))

(define win1 (and (heads) redball))
```

Church by example:

# A bit of gambling

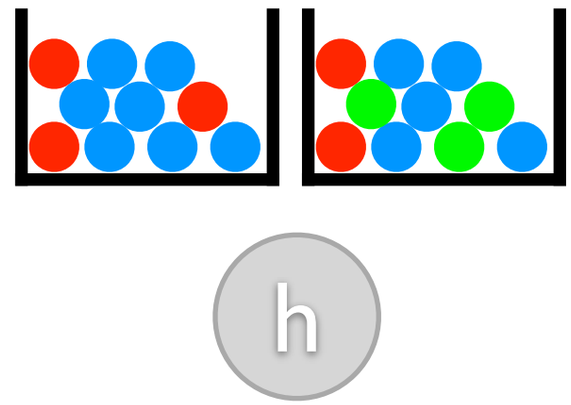



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                      (multinomial '(red green blue) '(0.2 0.3 0.5)))))

(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))

(define win1 (and (heads) redball))

(define win2 (equal? (color1) (color2)))
```

Church by example:

# A bit of gambling

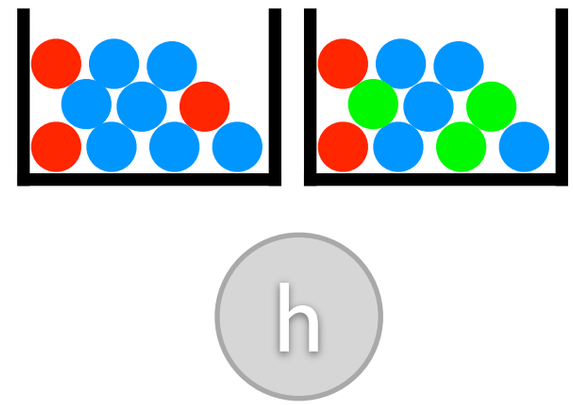



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                      (multinomial '(red green blue) '(0.2 0.3 0.5)))))

(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))

(define win1 (and (heads) redball))

(define win2 (equal? (color1) (color2)))
```

Church by example:

# A bit of gambling

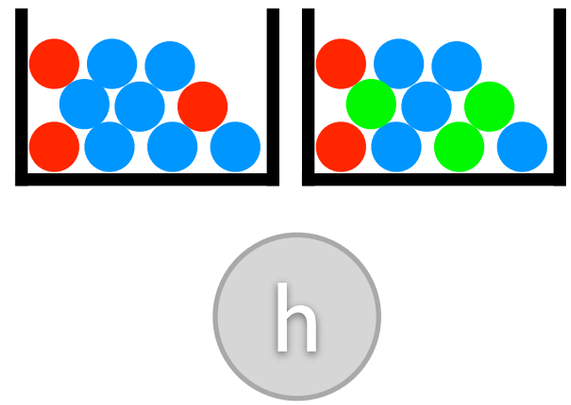



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                      (multinomial '(red green blue) '(0.2 0.3 0.5)))))

(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))

(define win1 (and (heads) redball))

(define win2 (equal? (color1) (color2)))

(define win (or win1 win2))
```

Church by example:

# A bit of gambling

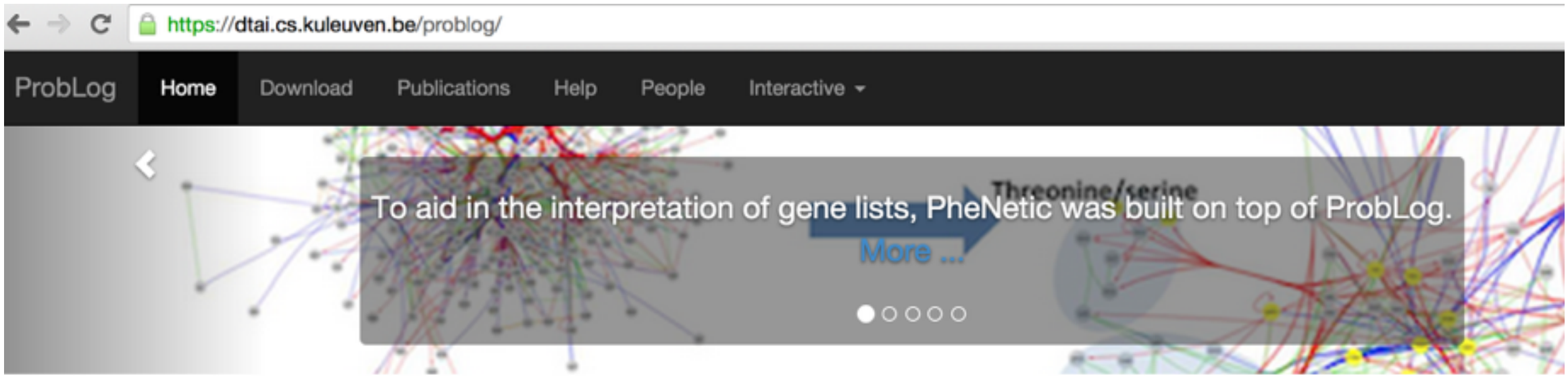



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                      (multinomial '(red green blue) '(0.2 0.3 0.5)))))

(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))

(define win1 (and (heads) redball))

(define win2 (equal? (color1) (color2)))

(define win (or win1 win2))
```

# Probabilistic Programming Summary

- Programming language + random primitives
- probabilistic generative model
- sampling or exact inference
- learning
- increasing number of probabilistic programming languages using various underlying paradigms
- Challenge : inference & scaling up …




# Introduction.

Probabilistic logic programs are logic programs in which some of the facts are annotated with probabilities.

ProbLog is a tool that allows you to intuitively build programs that do not only encode **complex interactions** between a large sets of **heterogenous components** bu **uncertainties** that are present in real-life situations.

The engine tackles several tasks such as computing the marginals given evidence and learning from (partial) interpretations. ProbLog is a suite of efficient algorithms tasks. It is based on a conversion of the program and the queries and evidence to a weighted Boolean formula. This allows us to reduce the inference tasks to well-s weighted model counting, which can be solved using state-of-the-art methods known from the graphical model and knowledge compilation literature.

# The Language. Probabilistic Logic Programming.

ProbLog makes it easy to express complex, probabilistic models.

```
0.3::stress(X) :- person(X).
0.2::influences(X,Y) :- person(X), person(Y).

smokes(X) :- stress(X).
smokes(X) :- friend(X,Y), influences(Y,X), smokes(Y).
```